# **Notas**

## **Conceptos**

**Expressions:** hace una acción y retorna un valor. Return X=1+2;

Expresiones Logicas: == > <

Expresiones primarias: ’Hola’ ,23 , true,

Expresiones de asignación: x=4

Var contador=1;

Contador`++; 🡺 retorna 2

**Statements:** hace una acción pero no retorna nada. declaraciones; ciclos , condicionales

Declaracion statements

1. Var prueba;
2. Function suma(a,b) {

//bloque de código

}

|  |  |
| --- | --- |
| Function Expressions  Const suma= function (a,b){ };  Array.map(function(){  } ); | Function Declarations  Function suma (a,b){ }; |

Conditional Statements: sirven para controlar el flujo.

If(condicion){}else {}

Bucles while(i<10){console.log(i)}

**Expressones Statements**

***En donde sea que el interprete espera un statements , nosotros podemos pasarle una expresion. “Expresiones Statatemens” . Pero esto no funciona en sentido inverso,***

## **JSI**

JAVASCRIPT: LENGUAJE INTERPRETADO

Lenguaje creado para ser usado en el front-end . Actualmente usa el motor V8 de google que mejora su velocidad.

Existe NodeJs que usa Js en el servidor. No es un lenguaje Tipado

**REPL: red evaluate print loop (Lee - Evalua - Imprime - Comienza de nuevo)**

**consola que sirve para ejecutar código y devuelve**

**Variable (var – let -const)**

Espacio reservado de memoria para poder almacenar algún valor

**Tipos de Datos**

* + Strings: cadena de caracteres
  + Numbers:
  + Booleans

|  |  |
| --- | --- |
| Null: no hay nada  Undefined: variable declarada pero no definida.  NaN: no sabe como interpretar  SCOPE  Objetos globales y locales: los objetos globales pueden usarse en cualquier lado del scope , en cambio los locales únicamente dentro de la función donde fueron delcarados. | Operadores  + - \* / = % ; el MOD (%)DEVUELVE EL RESTO EN UNA DIVISION |

**Funciones y sus argumentos**

Permite realizar una operación o caja donde la podre ejecutaro invocar n veces. Y me devolverá un resultado.

Los Argumentos son los valores que se pasan por parámetro a la función .

* function sumar(a,b){return a+b};
* sumar2=function(a,b){return a+b};
* const sumar=Function(a,b){return a+b};
* const sumar=(a,b)=> a+b;
* ojo con esto 🡺 const sumar=(a,b)=> {return a+b}; //si le pongo llaves debo poner el return

**Funciones de Javascript (librerias)**

* Math.pow()
* Math.round(): redondeara al mas cercano
* Math.floor():redondeara al numero mas cercano para abjo
* Math.ceil():redondeara al numero mas cercano para arriba
* .length()

**Control de flujo**

**const p=(x)=>**

**{**

**if (x>18) {console.log("Es mayor")**

**} else {console.log("Es menor")}**

**};**

## **JSll**

**Operadores estrictos**

**1==”1” : esto devuelve true, porque convierte el 1 numero a string . JS compara tipo de datos num y string haciendo un casting**

**1===”1” aca devuelve false por que compara estrictamente el tipo de Dato.**

**Operador Logico**

**Or: ||**

**And: &&**

**NOT : ¡==**

**Bucles**

|  |  |  |
| --- | --- | --- |
| **For** **for (let index = 0; index < 10; index++) {**  **console.log("valor: " + index);**  **}**  **valor: 0**  **valor: 1**  **valor: 2**  **…..**  **valor:9** | **const n=10;**  **let i=0;**  **while(i < 10){**  **console.log("print:" + i);**  **i++;**  **}**  **valor: 0**  **valor: 1**  **valor: 2**  **…..**  **valor:9** | **do{**  **console.log("Rocio");**  **i++;**  **}while(i<20);** |

#### Args();

**function args(){ console.log(arguments);}**

## **JSlll**

### **Arrays**

Un array o vector es una colección de datos.

**Declaracion**

const vector=[];

const numeros=[1,3,6,7,9];

Acceso números[0]; números[1];

números.length();

**Asignar Valores o agregar PUSH()**

Números[0]=”lo piso”;

Números.push(1);//lo agrega al ultimo

Var str=números[2];

**Pop() //saca el valor de la lsta**

Var nuevalista=números.pop();

TAMBIEN tenemos **shiff() y unShift() cambia en que operan con el primer elemento del array**

* **Unshift : agrega al primer elemento del array**
* **shift : saca el primer elemento**

**Arguments**

function multiplicarArgumentos() {

    // Usa la palabra clave `arguments` para multiplicar todos los argumentos y devolver el producto

    // Si no se pasan argumentos devuelve 0. Si se pasa un argumento, simplemente devuélvelo

    // Escribe tu código aquí:

    if (arguments.length < 1) return 0;

    var total = 1;

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

        total = total \* arguments[i];

    }

    return total;

}

**Recorrer Elementos**

**For Loop**

function cuentoElementos(arreglo) {

    //Realiza una función que retorne la cantidad de los elementos del arreglo cuyo valor es mayor a 18.

    //Escribe tu código aquí

    let contador = 0;

    for (let index = 0; index < arreglo.length; index++) {

        if (arreglo[index] > 18) {

            contador++;

        }

    }

    return contador;

}

## **JSIV**

Que es un keywords en javascript o en otros lenguajes?

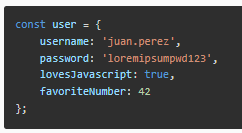
Las palabras clave son palabras reservadas en JavaScript que no puede usar para indicar etiquetas de variables o nombres de funciones

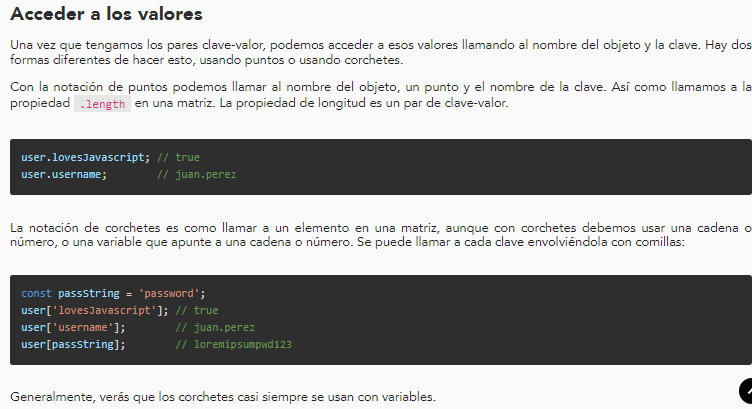
**Objetos**

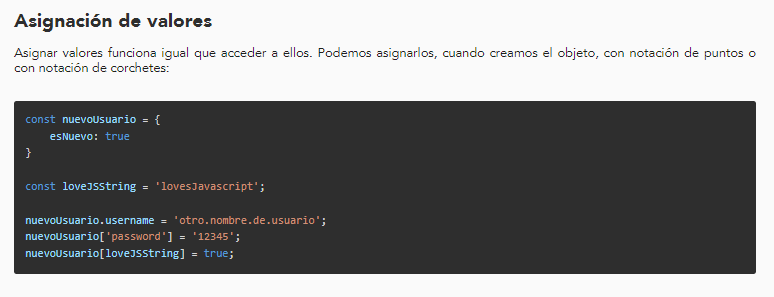
Un objeto es un conjunto de elementos o datos relacionados uno con los otros. USANDO LA ABSTRACCION.

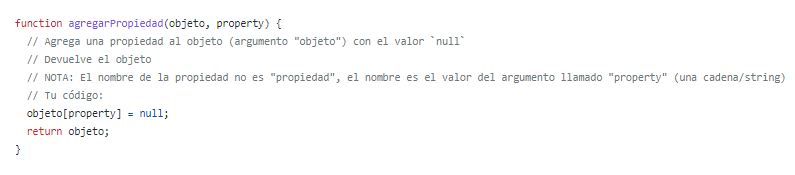
Par(clave:valor)

Const <nombreObjeto>={clave:valor}

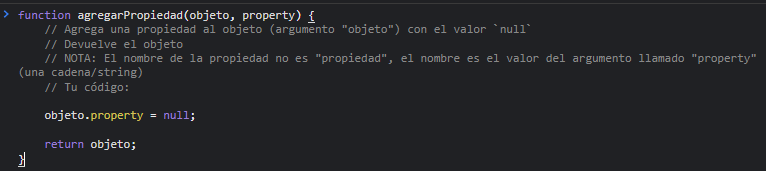




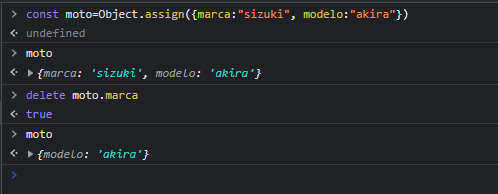


****

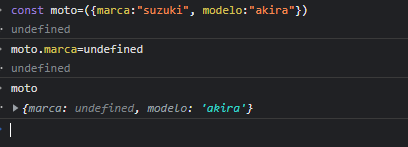
**Una variación de asignación**

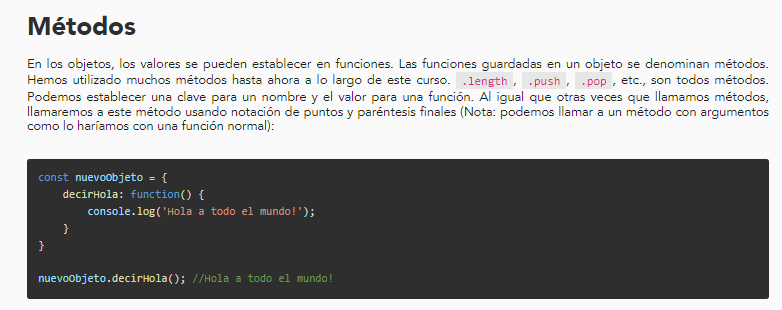
****

**Eliminar Propiedad usando la palabra clave DELETE.**

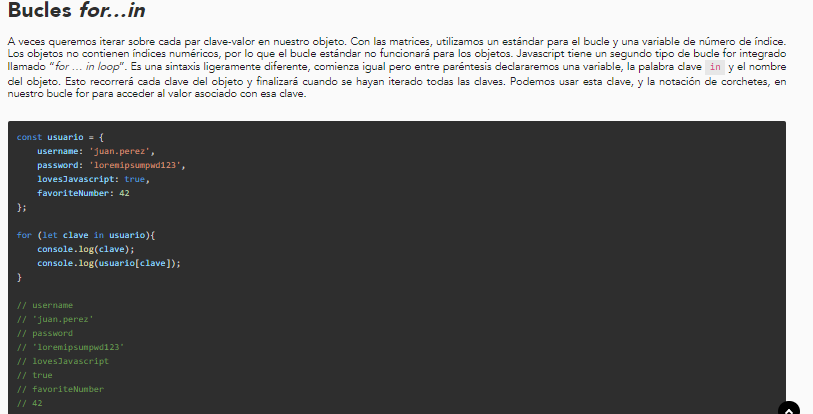
****

****

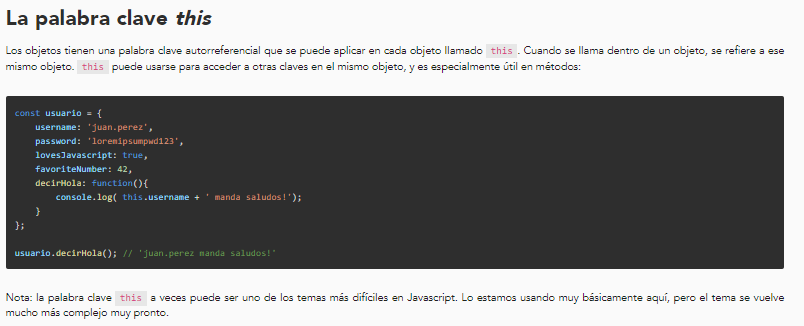
****

****

## For in es como el foreach

****

## Keyword this.

****

**El this depende mucho del contexto donde se este ejecutando , puede ser un método de un objeto , en el ámbito global o el de una función local .**

## **JSV**

### **Clases: es una plantilla para crear objetos, una plantilla.**

Nota: javascript no es un lenguaje puramente para objetos , en cambio java si. Js es prototipado.

Objeto

Const obj={ clave:valor};

* Keyword new
* Function contructora
* This

**Clase**

Function Persona(){

nombre:””,

apellido:””,

dni:””,

cuit:0;

correo:””,

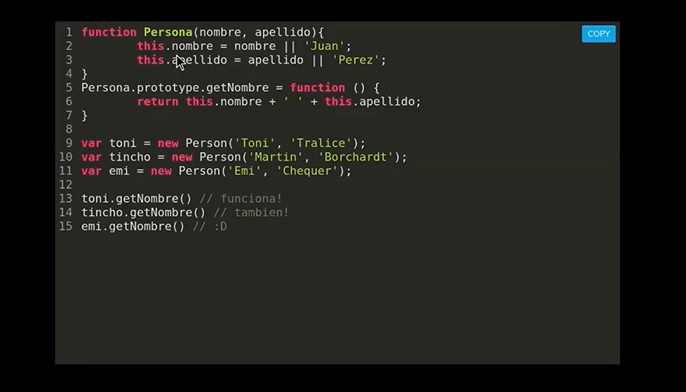
};

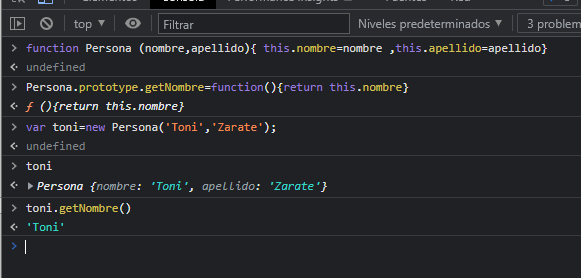
El keyword New crea un Objeto Ariel de tipo Persona.

Const Ariel= new Persona(“Ariel”, “Zarate”, “32785551”,20327855515,”arieltecnico@gmail.com” );

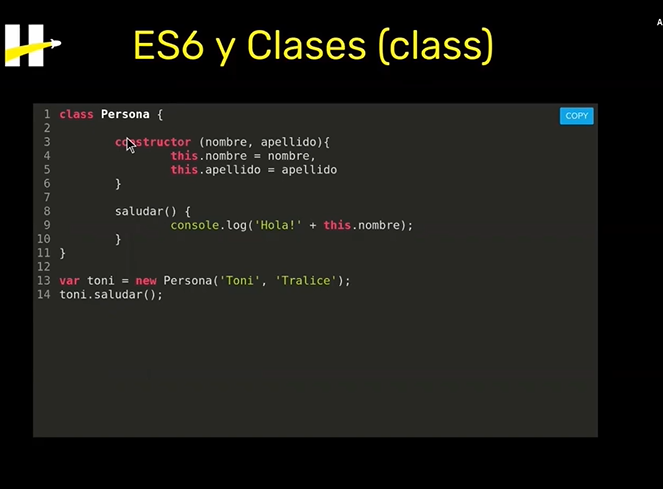
### Prototipos



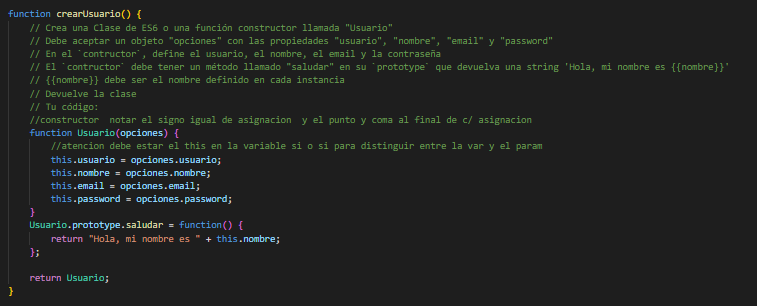
****

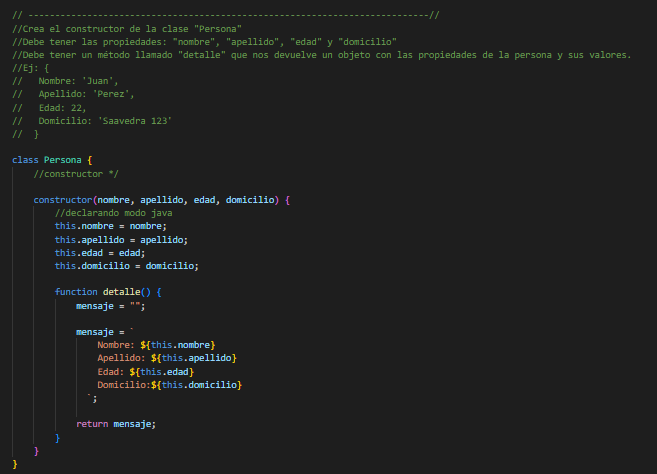
****

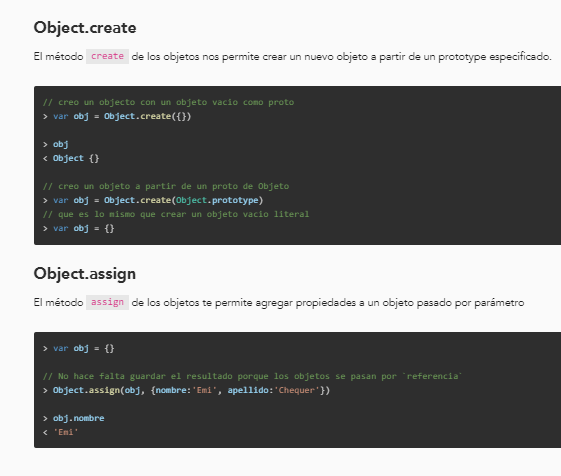




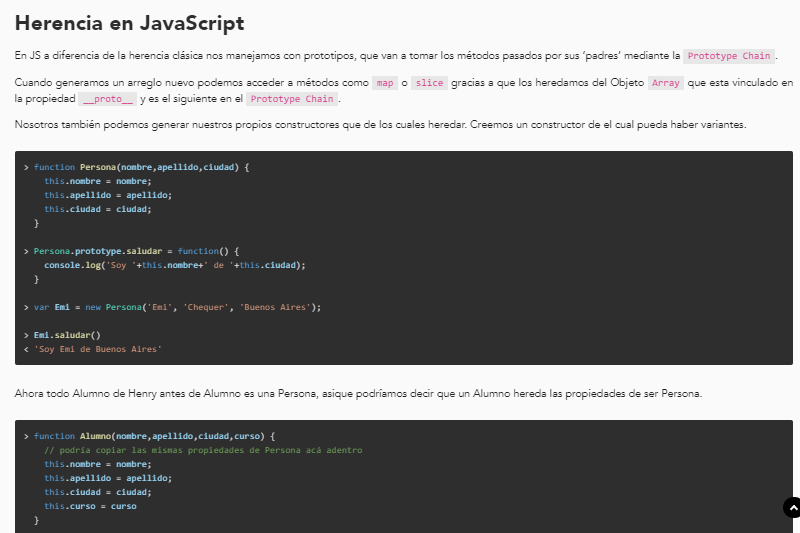
#### **Ejemplos en vivo**







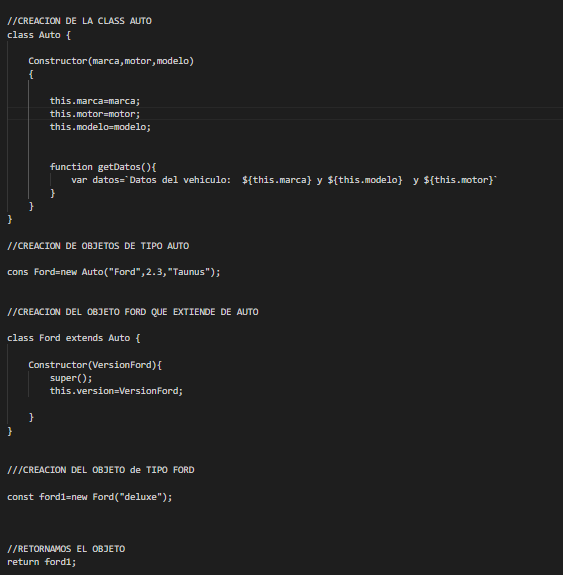
## Herencia prototype;

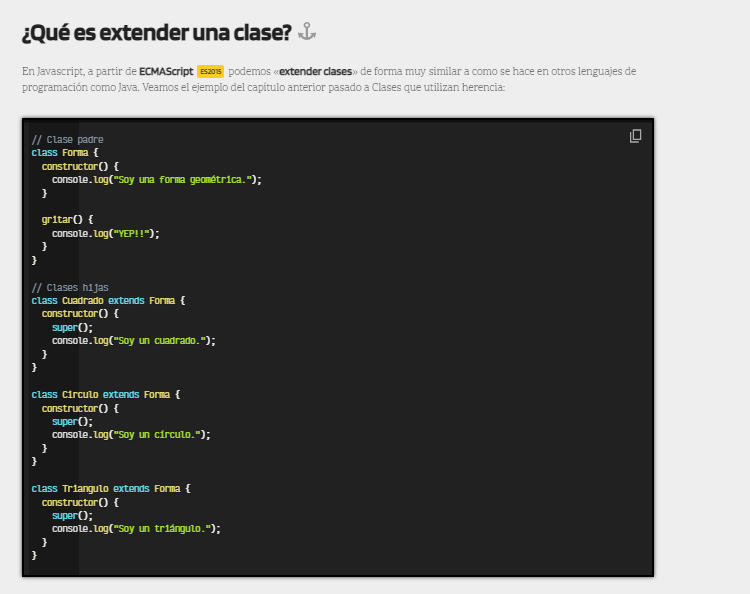


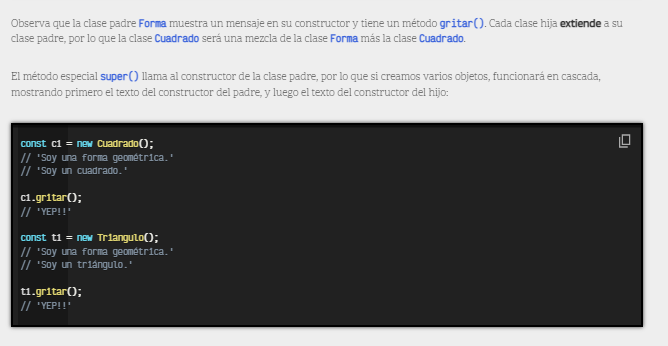
## **poo**

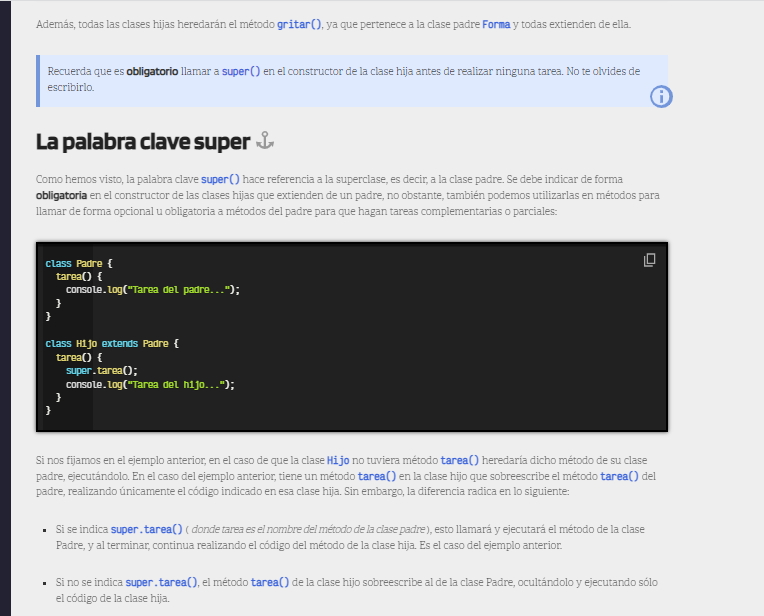
***Las características del POO son : Herencia, polimorfismo, Abstracción , Encapsulamiento. Modularidad.***

#### **Herencia mediante EMC6**









#### Callback

es una función pasada por argumento de otra función o como argumento





#### Definición y uso

El reduce()método ejecuta una función reductora para el elemento de matriz.

El reduce()método devuelve un único valor: el resultado acumulado de la función.

El reduce()método no ejecuta la función para elementos de matriz vacíos.

El reduce()método no cambia la matriz original.

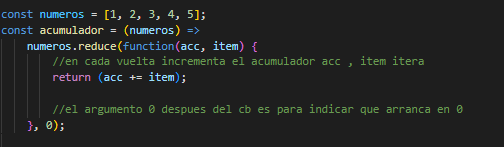
La función **reductora** recibe cuatro argumentos:

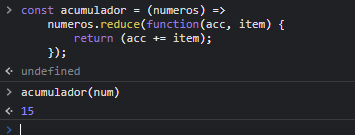
1. acumulador ( acc)
2. Valor Real ( cur)
3. Índice Real ( *idx*)
4. matriz ( *src*)

El valor devuelto de la función **reductora** se asigna al acumulador, cuyo valor se recuerda en cada iteración de la matriz y, en última instancia, se convierte en el valor final, único y resultante.

## [Sintaxis](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Array/Reduce#sintaxis)

**arr.reduce(callback(acumulador, valorActual[, índice[, array]])[, valorInicial])**

****

****

## Agregar elementos usando splice()

splice()puede tomar hasta 3 parámetros, pero por ahora, nos centraremos solo en los primeros 2. Los dos primeros parámetros de splice()son números enteros que representan índices o posiciones de elementos en la matriz a la que splice()se está llamando. Y recuerde, los arreglos tienen *un índice cero* , por lo que para indicar el primer elemento de un arreglo, usaríamos 0. splice()El primer parámetro de representa el índice en la matriz desde la que comenzar a eliminar elementos, mientras que el segundo parámetro indica la cantidad de elementos que se eliminarán. Por ejemplo:

let array = ['today', 'was', 'not', 'so', 'great'];

array.splice(2, 2);

Aquí eliminamos 2 elementos, comenzando con el tercer elemento (en el índice 2). arraytendría el valor ['today', 'was', 'great'].

splice()no solo modifica la matriz a la que se llama, sino que también devuelve una nueva matriz que contiene el valor de los elementos eliminados:

let array = ['I', 'am', 'feeling', 'really', 'happy'];

let newArray = array.splice(3, 2);

newArraytiene el valor ['really', 'happy'].

¿Recuerdas en el último desafío que mencionamos que splice()puede tomar hasta tres parámetros? Bueno, puede usar el tercer parámetro, compuesto por uno o más elementos, para agregarlo a la matriz. Esto puede ser increíblemente útil para cambiar rápidamente un elemento, o un conjunto de elementos, por otro.

const numbers = [10, 11, 12, 12, 15];

const startIndex = 3;

const amountToDelete = 1;

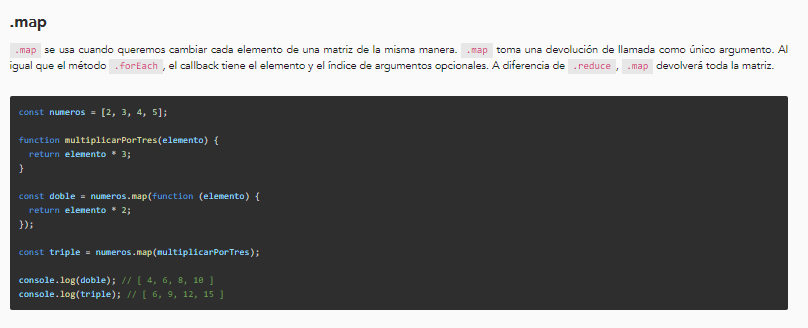
numbers.splice(startIndex, amountToDelete, 13, 14);

console.log(numbers);

La segunda aparición de 12se elimina y agregamos 13y 14en el mismo índice. La numbersmatriz ahora sería [ 10, 11, 12, 13, 14, 15 ].

Aquí, comenzamos con una matriz de números. Luego, le pasamos lo siguiente a splice(): El índice en el que comenzar a eliminar elementos (3), el número de elementos a eliminar (1) y los argumentos restantes (13, 14) se insertarán a partir de ese mismo índice. Tenga en cuenta que puede haber cualquier número de elementos (separados por comas) después amountToDeletede , cada uno de los cuales se inserta.

## Map



### forEach

**//array de objetos**

let carros = [

{ "color": "morado",

"tipo": "minivan",

"registroDia": new Date('2017-01-03'),

"capacidad": 7

},

{ "color": "rojo",

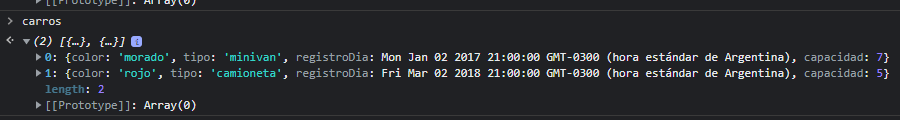
"tipo": "camioneta",

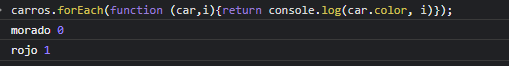
"registroDia": new Date('2018-03-03'),

"capacidad": 5

}

]





## Slice()

Copiar elementos de matriz usando slice()

El siguiente método que cubriremos es slice(). En lugar de modificar una matriz, slice()copia o extrae una cantidad dada de elementos a una nueva matriz, dejando intacta la matriz a la que se llama. slice()toma solo 2 parámetros: el primero es el índice en el que comienza la extracción y el segundo es el índice en el que se detiene la extracción (la extracción ocurrirá hasta el elemento en este índice, pero sin incluirlo). Considera esto:

let weatherConditions = ['rain', 'snow', 'sleet', 'hail', 'clear'];

let todaysWeather = weatherConditions.slice(1, 3);

todaysWeathertendría el valor ['snow', 'sleet'], mientras weatherConditionsque todavía tendría ['rain', 'snow', 'sleet', 'hail', 'clear'].

En efecto, hemos creado una nueva matriz extrayendo elementos de una matriz existente.

Hemos definido una función, forecast, que toma una matriz como argumento. Modifique la función usando slice()para extraer información de la matriz de argumentos y devolver una nueva matriz que contenga los elementos de cadena warmy sunny.

function forecast(arr) {

  // Only change code below this line

let nuevoArray=arr.slice(2,4);

  return nuevoArray;

}

// Only change code above this line

console.log(forecast(['cold', 'rainy', 'warm', 'sunny', 'cool', 'thunderstorms']));

## Operador spreed . . .

Copie una matriz con el operador de propagación

Si bien slice()nos permite ser selectivos sobre qué elementos de una matriz copiar, entre varias otras tareas útiles, el nuevo operador de extensión de ES6 nos permite copiar fácilmente todos los elementos de una matriz, en orden, con una sintaxis simple y fácil de leer. La sintaxis extendida simplemente se ve así:...

En la práctica, podemos usar el operador de propagación para copiar una matriz de esta manera:

let thisArray = [true, true, undefined, false, null];

let thatArray = [...thisArray];

thatArray[true, true, undefined, false, null] es igual thisArraypermanece sin cambios y thatArraycontiene los mismos elementos que thisArray.

Comprobar la presencia de un elemento con indexOf()

Dado que las matrices se pueden cambiar o *mutar* en cualquier momento, no hay garantía sobre dónde estará una parte de los datos en particular en una matriz determinada, o si ese elemento aún existe. Afortunadamente, JavaScript nos brinda otro método incorporado indexOf(), que nos permite verificar rápida y fácilmente la presencia de un elemento en una matriz. indexOf()toma un elemento como parámetro y, cuando se llama, devuelve la posición o el índice de ese elemento, o -1si el elemento no existe en la matriz.

Por ejemplo:

let fruits = ['apples', 'pears', 'oranges', 'peaches', 'pears'];

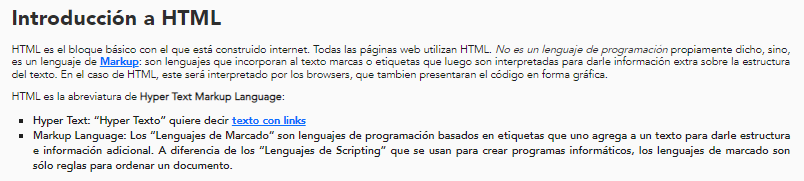
fruits.indexOf('dates');

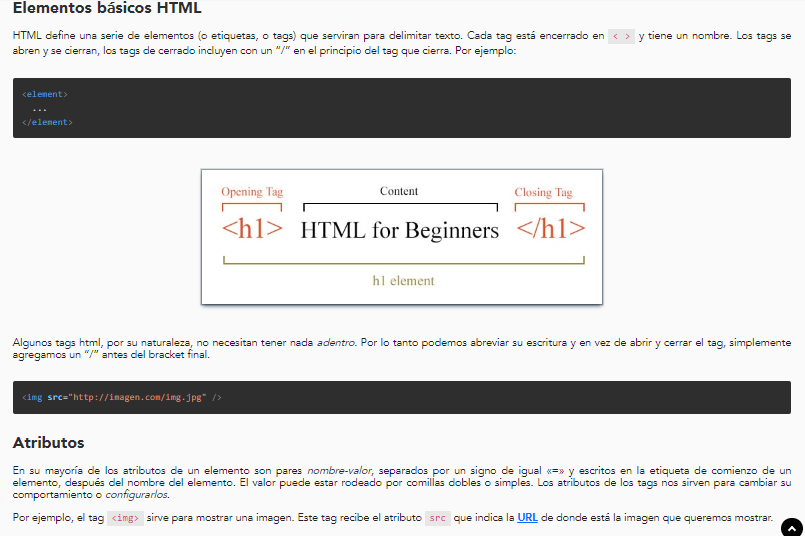
fruits.indexOf('oranges');

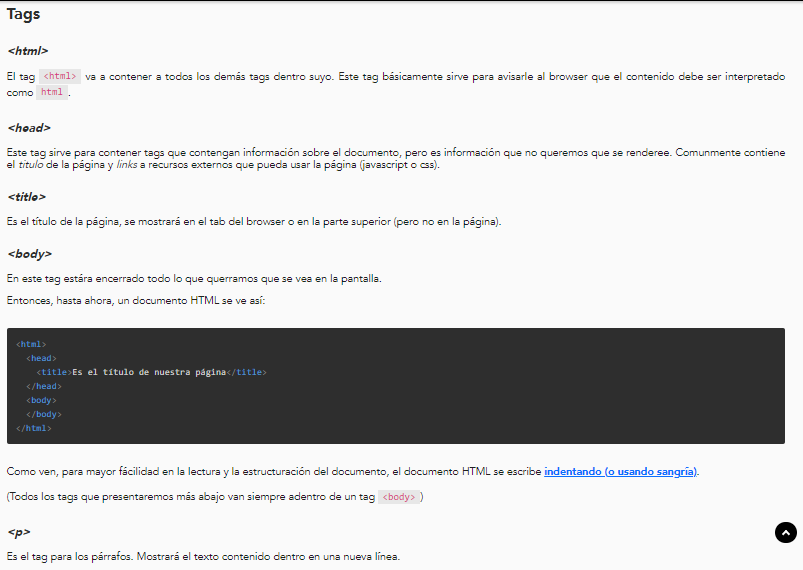
fruits.indexOf('pears');

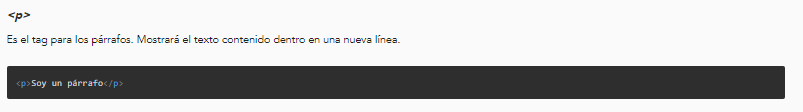
indexOf('dates')devuelve -1, indexOf('oranges')devuelve 2y indexOf('pears')devuelve 1(el primer índice en el que existe cada elemento).

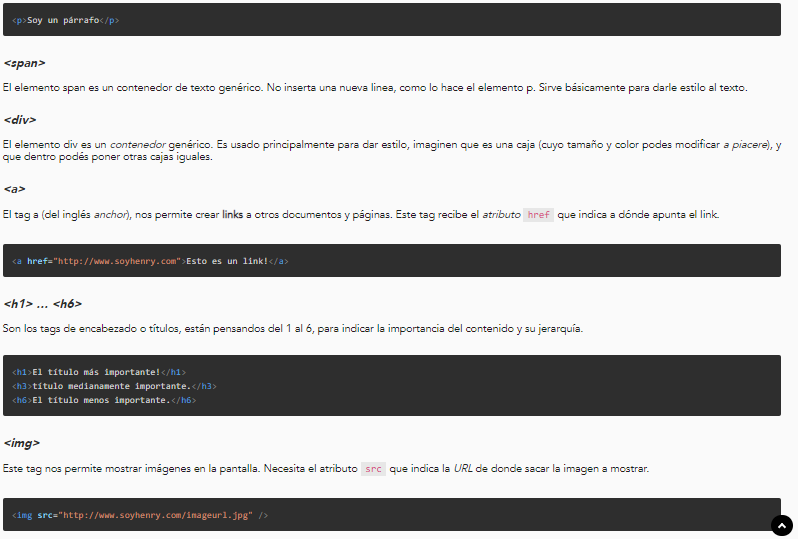
## HTML











## **CSS**

