

# CARACTERÍSTICAS Y GRAMÁTICA

## Isomorfismo

Hoy JavaScript, es el único lenguaje capaz de ejecutarse en las 3 capas de una aplicación

1. Frontend (JavaScript)
2. Backend (JavaScript)
3. Persistencia de Datos (MongoDB, Couch DB, Firebase, etc).

---

## Con JavaScript puedes:

- Diseño y desarrollo web.
- Hacer videojuegos
- Experiencias 3D, Realidad aumentada, realidad virtual.
- Aplicaciones híbridas y móviles.
- Machine Learning.

---

## Características:

- Lenguaje de alto nivel.
- Interpretado.
- Dinámico.
- Débilmente tipado.
- Multi paradigma.
- Sensible a mayúsculas y minúsculas.
- No necesitas los puntos y comas al final de cada línea.

---

## Escritura de código:

Los identificadores deben comenzar con:

- Una letra o
- Un signo de dolar \$ o
- Un guión bajo \_
- Nunca con números o caracteres especiales.

---

## Usa snake\_case en: (separado por guion bajo)

En nombre de archivos:

```
mi_archivo_javascript.js;
```

---

## Usa UPPER\_CASE en: (todo mayuscula)

En constantes:

```
const UNA_CONSTANTE = "Soy una constante",  
PI = 3.141592653589793;
```

## Usa UpperCamelCase en: (primera de cada palabra en mayúsculas)

En clases:

```
class SerHumano {  
    constructor(nombre, genero) {  
        this.nombre = nombre;  
        this.genero = genero;  
    }  
  
    miNombreEs() {  
        return `Mi nombre es ${this.nombre}`;  
    }  
}
```

---

## Usa lowerCamelCase en: (inicia en minúsculas, y demás palabras en mayuscula)

Objetos:

```
const unObjeto = {  
    nombre: "Jonathan",  
    email: "jonmircha@gmail.com",  
};
```

Primitivos:

```
let unaCadena = "Hola Mundo",  
    unNumero = 19,  
    unBoolean = true;
```

Funciones:

```
function holaMundo(nombre) {  
    alert(`Hola mundo ${nombre}`);  
}  
holaMundo("Jonathan");
```

Instancias:

```
const ajax = new XMLHttpRequest(),  
    jon = new SerHumano("Jonathan", "Hombre");
```

## Palabras reservadas

**A:** abstract  
**B:** boolean, `break`, byte  
**C:** `case`, catch, char, `class`, `const`, `continue`  
**D:** `debugger`, `default`, `delete`, `do`, double  
**E:** `else`, `enum`, `export`, `extends`  
**F:** `false`, final, finally, float, `for`, `function`  
**G:** goto  
**I:** `if`, `implements`, `import`, `in`, `instanceof`, `int`, `interface`  
**L:** `let`, long  
**N:** native, `new`, `null`  
**P:** `package`, `private`, `protected`, `public`  
**R:** `return`  
**S:** short, `static`, `super`, `switch`, `synchronized`  
**T:** `this`, `throw`, `throws`, `transient`, `true`, `try`, `typeof`  
**V:** `var`, `volatile`, `void`  
**W:** `while`, `with`

---

## Ordenamiento de código

1. IMPORTACIÓN DE MÓDULOS.
2. DECLARACIÓN DE VARIABLES.
3. DECLARACIÓN DE FUNCIONES.
4. EJECUCIÓN DE CÓDIGO.

---

## Tipos de datos en JavaScript

Primitivos: Se accede directamente al valor.

- string
- number
- boolean
- null
- undefined
- NaN


Compuestos: Se accede a la referencia del valor.

- object = {}
- array = []
- function () { }
- Class {}
- etc.

## Variables var, let

- No es muy recomendable usar var para las variables.
- Las variables var serán de ámbito global.
- Las variables let serán de ámbito de bloque. (dentro de if por ejemplo)

Con var:



```
var musica = "Rock";
console.log("Variable Música antes del Bloque", musica);
{
  var musica = "Pop";
  console.log("Variable Música dentro del Bloque", musica);
}
console.log("Variable Música después del Bloque", musica);
```

Variable Música antes del Bloque  
Rock

Variable Música dentro del Bloque  
Pop

Variable Música después del Bloque  
Pop

Con let:



```
let musica2 = "Rock";
console.log("Variable Música antes del Bloque", musica2);
{
  let musica2 = "Pop";
  console.log("Variable Música dentro del Bloque", musica2);
}
console.log("Variable Música después del Bloque", musica2);
```

Variable Música antes del Bloque  
Rock

Variable Música dentro del Bloque  
Pop

Variable Música después del Bloque  
Rock

## Variable const

- Son variables que no cambian valor.
- Hay que inicializarlas.
- Se usan cuando no quiero que cambie de valor en el flujo del programa.
- Cambia cuando se usa en tipo de datos compuestos (objetos, array, etc) ya que no se accede a tal variable sino que a una referencia.

## Cadenas de texto (Strings)

**Tamaño:** nombre.length

**Mayúscula:** nombre.toUpperCase()

**Minúscula:** nombre.toLowerCase()

**Buscar:** nombre.includes("textoABuscar") // Retorna true o false

**Quitar espacios a los lados:** nombre.trim()

**Convertir a array:** nombre.split(",") // Separa el texto por comas y lo guarda en un array

# Template Strings

Interpolación de variables, tildes invertidas:

```
let nombre = "Christopher";
let apellido = "Gudiel";
let saludo = `Mi nombre es: ${nombre} ${apellido}`;
console.log(saludo);
```

# Numeros (Numbers)

Mostrar cierta cantidad de decimales: numero.toFixed(2);

Parte entera de un numero: parseInt(numero);

Parte decimal de un numero: parseFloat(numero);

Tipo de dato: typeof variable

# Booleanos

True	False
<pre>if (true) if ({}) if ([]) if (42) if ("foo") if (new Date()) if (-42) if (3.14) if (-3.14) if (Infinity) if (-Infinity)</pre>	<pre>if (false) if (null) if (undefined) if (0) if (-0) if (0n) if (NaN) if ("")</pre>

===	true	false	0	"	null	undefined	NaN	Infinity	[]	{}
true	true	false	false	false	false	false	false	false	false	false
false	false	true	false	false	false	false	false	false	false	false
0	false	false	true	false	false	false	false	false	false	false
"	false	false	false	true	false	false	false	false	false	false
null	false	false	false	false	true	false	false	false	false	false
undefined	false	false	false	false	false	true	false	false	false	false
NaN	false	false	false	false	false	false	false	false	false	false
Infinity	false	false	false	false	false	false	false	true	false	false

===	true	false	0	"	null	undefined	NaN	Infinity	[]	{}
[]	false	false	false	false	false	false	false	false	false	false
{}	false	false	false	false	false	false	false	false	false	false

## Undefined, null & NaN

```
//undefined indica que no se ha
//inicializado una variable y que el valor
//está ausente
let indefinida;
console.log(indefinida);

//null es un valor especial que indica la
//ausencia de un valor
let nulo = null;
console.log(null);

//NaN - Not a Number
let noEsUnNumero = "hola" * 3.7;
console.log(noEsUnNumero);
```

## Funciones

```
//Funcion normal
function estaEsFuncion(){
  console.log("Hola mundo");
}

//Funcion que retorna un valor
function retornaAlgo(){
  return "Hola mundo 2";
}

//Funcion que recibe parametros
function conParametro(datos){
  console.log(datos);
}

estaEsFuncion();
console.log(retornaAlgo());
conParametro("Hola Mundo 3");
```

## Funciones declaradas

Se pueden invocar antes de crearlas.

```
estaEsFuncion();

function estaEsFuncion(){
  console.log("Hola mundo");
}
```

## Función expresada (anónima)

```
//Funcion anonima
const funcionExpresada = function() {
  console.log("Hola Mundo 4");
}

funcionExpresada();
```

## Arreglos (Arrays)

```
const a = [];
const b = [1, true, "Hola", ["A","B","C", [1,2,3]]];

console.log(a);
console.log(b);
console.log(b.length); //Tamano de array
console.log(b[3][2]); // Imprimiendo la letra C
console.log(b[3][3][0]); // Imprimiendo el numero 1

const c = Array.of("x","y","z");
console.log(c);

//Array de tamaño 10, que se llenan con 'false'
const d = Array(10).fill(false);
console.log(d);

//ya no se usa mucho
const e = new Array();
console.log(e);

//ya no se usa mucho
const f = new Array(1,2,3);
console.log(f);

const colores = ["gris", "negro", "verde"];
console.log(colores);

//Agrega al final
colores.push("Negro");
console.log(colores);
```

```
//Quita el ultimo
colores.pop();
console.log(colores);

//recorriendo arreglo
colores.forEach(function(elemento, index){
    console.log(index, elemento);
});
```

## Objetos

```
let a = new String("Hola mundo");

const b = {};

const c = new Object();

const gudiel = {
    nombre: "Christopher",
    apellido: "Gudiel",
    edad: "26",
    casado: false,
    contacto: {
        email: "gudiel.lv16@gmail.com",
        movil: "12345678"
    },
    saludar: function(){
        console.log("Hola mundo")
    },
    saludar2: function(){
        //this hace referencia a variables del mismo objeto
        console.log(`Mi nombre es: ${this.nombre} ${this.apellido}`)
    }
}

console.log(gudiel);
console.log(gudiel["nombre"]);
console.log(gudiel.contacto.email);
gudiel.saludar(); //imprime de una vez, ya que ejecuta la funcion
gudiel.saludar2();

console.log(Object.keys(gudiel)); //lista llaves del array
console.log(Object.values(gudiel)); //lista valores del array
console.log(gudiel.hasOwnProperty("nombre")); //permite saber si tiene alguna propiedad
(true|false)
```



## Tipos de operadores

```
let a = 5 + (5-10)*3;

let modulo = 5%2;

console.log(a);
console.log(modulo);

/*Relacionales >, <, >=, <=, ==, ===, !=, !==*/

console.log(8>9); //false
console.log(8<9); //true
console.log(8>=9); //false
console.log(10>=9); //true

console.log("-----");
/*
= es asignacion de variable
== es comparacion de valores
=== es comparacion de valores y tipo de variable
*/

console.log(7==7); //true
console.log("7"==7); //true
console.log("7"===7); //false

console.log("-----");
/*Incremento, Decremento */
let i = 1;
i = i +2;
i += 3;
i -= 2;
i /= 2;
i *= 5;
console.log(i);

console.log("-----");
/*operador unario, suma aunque este dentro de un console.log() */
i++;
i--;
++i;
--i;
console.log(i);

console.log("-----");
/*operadores logicos*/
/*
! - Not, niega, lo que es verdadero lo vuelve falso y viceversa
|| - Or, con que una se cumpla, el Or validara
&& - And, Las dos se cumplen
*/
```

```
*/  
  
console.log(!true); //false  
console.log(!false); //true  
console.log((9===9)||("9"===9)) //true  
console.log(((9===9)&&("9"===9))) //false
```

## Condicionales

```
//COMUN  
let edad = 18;  
  
if (edad>18){  
    console.log("Mayor de edad")  
}  
else if (edad==18){  
    console.log("Justo con la edad")  
}else{  
    console.log("Menor de edad")  
}  
  
//OPERADOR TERNARIO  
console.log("OPERADOR TERNARIO");  
let num = 10;  
  
let eresMayor = (num >= 10)  
    ? "TRUE"  
    : "FALSE";  
  
console.log(eresMayor);  
  
//SWITCH - CASE  
console.log("SWITCH - CASE");  
  
let dia = 1;  
switch(dia){  
    case 0:  
        console.log("Viernes");  
        break;  
    case 1:  
        console.log("Sabado");  
        break;  
    default:  
        console.log("Default");  
        break;  
}
```

## Ciclos (Loops)

```
let contador = 0;

//WHILE
while(contador<5){
    console.log("while",contador);
    contador++;
}

//DO WHILE
do{
    console.log("do while",contador);
    contador++
}while(contador<8);

//FOR
for (let i = 0; i < 5; i++) {
    console.log("for",i);
}

//FOR
let numeros = [1,2,3,4,5,6,7,8,9];
for (let i = 0; i < numeros.length; i++) {
    console.log("i en array: ",numeros[i]);
}

//FOR IN
const gudiel = {
    nombre: "Christopher",
    apellido: "Gudiel"
}
for (const key in gudiel) {
    console.log(key,gudiel[key]);
}

//FOR OF (es mas para arreglos o objetos iterables en js)
let cadena = "Hola mundo";
for (const caracter of cadena) {
    console.log(caracter);
}
```

## Manejo de errores (Try-Catch)

```
try {
  console.log("En el Try se agrega el codigo a evaluar");
} catch (error) {
  console.log("Catch, captura cualquier error surgido o lanzado en el try");
} finally{
  console.log("Se ejecuta siempre al final de un bloque try-catch");
}

try {
  let numero = 10;
  if(isNaN(numero)){ //si es numero
    throw new Error("No es un numero");
  }
  console.log(numero*numero);
} catch (error) {
  console.log("Se produjo el siguiente error: ",error);
}
```

## Break & Continue

```
const numeros = [1,2,3,4,5,6,7,8,9];

console.log("Break");
for (let i = 0; i < numeros.length; i++) {
  if (i===5){
    break; //se sale del for
  }
  console.log(numeros[i]);
}

console.log("Continue");
for (let i = 0; i < numeros.length; i++) {
  if (i===5){
    continue; //no ejecuta lo que sigue despues de aca, y continua el for
              //en este caso no imprime el 6
  }
  console.log(numeros[i]);
}
```

## Desestructuración

```
const numeros = [1,2,3];

//sin desestructuración
let uno = numeros[0];
let dos = numeros[1];
let tres = numeros[2];
console.log(uno,dos,tres);

//Con desestructuración
const [one,two,three] = numeros;
console.log(one,two,three);

let persona = {
  nombre: "Christopher",
  apellido: "gudiel",
  edad: "26"
}

//Con desestructuración, en este caso, las variables tienen que tener
//el mismo nombre que en el objeto
let {nombre,apellido,edad} = persona;
console.log(nombre,apellido,edad);
```

## Objetos literales

```
let nombre = "cAaG", edad = 7;

const perro = {
  nombre: nombre,
  edad: edad,
  ladrar: function(){
    console.log("Guaa, Guaa");
  }
}

console.log(perro);
perro.ladrar();

const dog = {
  nombre, //se hace así cuando tienen el mismo nombre: ej. nombre: nombre
  edad,
  raza: "Callejero",
  ladrar(){
    console.log("Guaa guaa")
  }
}

console.log(dog);
```

## Parámetros REST, Operador Spread (operador de propagación)

```
//PARAMETROS REST
//Se crean dinamicamente las variables con los ...
//No se sabe cuantas variables se pueden recibir por eso el ...
//pueden venir solo dos variables (a y b), o pueden venir mas
function sumar(a,b, ...C){
  let resultado = a+b;
  C.forEach(function(n){
    resultado += n
  });

  return resultado;
}

console.log(sumar(1,2));
console.log(sumar(1,2,3));
console.log(sumar(1,2,3,4));

//OPERADOR SPREAD (operador de propagacion)

const arr1 = [1,2,3,4,5];
const arr2 = [6,7,8,9,0];
console.log(arr1,arr2);

//se vuelve un array de 10
const arr3 = [...arr1, ...arr2];
console.log(arr3);
```

## Arrow Functions (funciones de flecha)

```
//Normal
const saludo = function(){
  console.log("Hola mundo");
}

saludo();

//Arrow functions
const saludar = (nombre) => {
  console.log(`Hola ${nombre}`);
}

saludar("Gudiel");

//se puede hacer de la siguiente forma
const sumar = (a,b) => a+b;
const sumar2 = (a,b) => {return a+b};

console.log(sumar(1,1),sumar2(2,2));
```

```
//OTRO EJEMPLO:
const numeros = [1,2,3,4,5];

//forma normal
numeros.forEach(function(el,index){
    console.log(el,index);
});

//arrow functions
numeros.forEach((el,index) => {
    console.log(el,index);
});
```

## Prototipos

**Clases** – Modelo a seguir.

**Objetos** – Es una instancia de una clase.

**Atributos** – Característica o propiedad del objeto (variables de un objeto).

**Métodos** – Son las acciones que un objeto puede realizar (son funciones dentro de un objeto).

```
/*Version 1
function Animal(nombre, genero){
    //Atributos
    this.nombre = nombre;
    this.genero = genero;

    //Metodos
    this.sonar = function(){
        console.log("Hago sonidos");
    }

    this.saludar = function(){
        console.log(`Hola me llamo ${this.nombre}`);
    }
}
*/

/*Version 2
Asignamos los metodos al prototipo, no a la funcion como tal (no como version 1)
Esto para mejorar el rendimiento y espacio en memoria, ya que no se crean los metodos en cada
instancia*/
function Animal(nombre, genero){
    //Atributos
    this.nombre = nombre;
    this.genero = genero;
}
```

```
//Metodos agregados al prototipo de la funcion constructura
Animal.prototype.sonar = function(){
    console.log("Hago sonidos");
}

Animal.prototype.saludar = function(){
    console.log(`Hola me llamo ${this.nombre}`);
}

const chui = new Animal("Chui","Macho");
const chuia = new Animal("Chuia","Hembra");

console.log(chui);
console.log(chuia);

chui.sonar();
chuia.sonar();
chui.saludar();
chuia.saludar();
```

## Herencia Prototípica

```
/*Asignamos los metodos al prototipo, no a la funcion como tal (no como version 1)
Esto para mejorar el rendimiento y espacio en memoria, ya que no se crean los metodos en cada
instancia*/
function Animal(nombre, genero){
    //Atributos
    this.nombre = nombre;
    this.genero = genero;
}

//Metodos agregados al prototipo de la funcion constructura
Animal.prototype.sonar = function(){
    console.log("Hago sonidos");
}

Animal.prototype.saludar = function(){
    console.log(`Hola me llamo ${this.nombre}`);
}

//Herencia prototipica
function Perro(nombre,genero,tamano){
    this.super = Animal;
    this.super(nombre,genero);
    this.tamano = tamano;
}

//Perro esta heredando de animal
Perro.prototype = new Animal();
Perro.prototype.constructor = Perro;
```



```
//Sobreescritura de metodos del prototipo padre en el hijo
Perro.prototype.sonar = function(){
    console.log("Soy un perro y mi sonido es un ladrido")
}

Perro.prototype.ladrear = function(){
    console.log("Guaa Guaa")
}

const chui = new Perro("Chui","Macho","Mediano");
const chuia = new Animal("Chuia","Hembra");

console.log(chui);
console.log(chuia);

chui.sonar();
chuia.sonar();
chui.saludar();
chuia.saludar();
```

## Clases y Herencia

```
class Animal{
    //Constructor
    constructor(nombre,genero){
        this.nombre = nombre;
        this.genero = genero;
    }

    //Metodos
    sonar(){
        console.log("Hago sonidos");
    }

    saludar(){
        console.log(`Hola me llamo ${this.nombre}`);
    }
}

class Perro extends Animal{
    constructor(nombre,genero,tamano){
        super(nombre,genero); //hace referencia a la clase padre y se le pasan los parametros
        this.tamano = tamano;
    }

    sonar(){
        console.log("Hago sonidos");
    }

    ladrear(){
        console.log("Guaa Guaa")
    }
}
```

```

    }
}

const chui = new Animal("Chui","Macho");
const chuia = new Perro("Chuia","Hembra","Gigante");

chui.sonar();
chuia.sonar();
chui.saludar();
chuia.saludar();
chuia.ladrrar();

```

## Métodos estáticos, getters y setters

```

class Animal{
    //Constructor
    constructor(nombre,genero){
        this.nombre = nombre;
        this.genero = genero;
    }

    //Metodos
    sonar(){
        console.log("Hago sonidos");
    }

    saludar(){
        console.log(`Hola me llamo ${this.nombre}`);
    }
}

class Perro extends Animal{
    constructor(nombre,genero,tamano){
        super(nombre,genero); //hace referencia a la clase padre y se le pasan los parametros
        this.tamano = tamano;
        this.raza = null;
    }

    sonar(){
        console.log("Hago sonidos");
    }

    ladrrar(){
        console.log("Guaa Guaa")
    }

    //Un metodo estatico se pueden ejecutar sin necesidad de instanciar la clase
    static queEres(){
        console.log("Un perro static")
    }
}

```

```

//Los setters y getters son metodos especiales que nos permiten establecer
//y obtener los valores de atributos de nuestra clase
get getRaza(){
    return this.raza;
}

set setRaza(raza){
    this.raza = raza;
}
}

Perro.queEres();

const chui = new Animal("Chui","Macho");
const chuia = new Perro("Chuia","Hembra","Gigante");

chui.sonar();
chuia.sonar();
chui.saludar();
chuia.saludar();
chuia.ladRAR();

//Se les envia y se obtienen como propiedad (sin parentesis)
chuia.setRaza = "Gran Danes";
console.log(chuia.getRaza);

```

## Objeto Console

```

console.log(console);
console.error("Esto es un error");
console.warn("Esto es un aviso");
console.info("Esto es un mensaje informativo");
console.log("Registro de lo que a pasado en nuestra app");
//Con un espacio:
console.log("a","b","c");
//Interpolando:
let nombre = "Gudiel";
console.log(`Mi nombre es: ${nombre}`);
//Comodines: (sustituye por las variables)
let edad=26;
console.log("Mi nombre es %s y tengo %d anios",nombre,edad);
//Limpiar consola:
//console.clear();

//console.log(document); //Muestrea todo el html

//muestra propiedades a detalle:
//console.dir(window);
//console.dir(document);

```

```
//Agrupando: .groupCollapse hace lo mismo
console.group("Cursos de Progra");
console.log("JavaScript");
console.log("Node js");
console.log("Angular");
console.groupEnd();

//Tabla:
console.table(Object.entries(console).sort()); //sort para ordenar

//Representar array en tabla:
const numeros = [1,2,3,4,5];
const vocales = ["a","e","i","o","u"];
console.table(numeros);
console.table(vocales);

const perro = {
  nombre: "a",
  raza: "chig",
  color: "blanco"
}

console.table(perro);

//Tiempo en que ejecuta (tiene que tener la misma bancera, lo que esta en comillas)
console.time("Cuanto tiempo tarda mi codigo");
const arreglo = Array(1000000);
for (let i = 0; i < arreglo.length; i++) {
  arreglo[i] = i;
}
console.timeEnd("Cuanto tiempo tarda mi codigo");

//Cuantas veces se ejecuta un fragmento de codigo
for (let i = 0; i < 3; i++) {
  console.count("codigo for");
  console.log(i);
}

//Mini lebreria para hacer pruebas
let x = 1, y = 2;
pruebaXY = "Se espera que x sea mayor que y";
console.assert(x<y,{x,y,pruebaXY});
```

## Objeto Date

```
console.log(Date());
let fecha = new Date();
//Dia del mes
console.log(fecha.getDate());
//Dia de la semana, Domingo a sabado [0,6]
console.log(fecha.getDay());
//Numero de mes, de enero a dic [0,11]
console.log(fecha.getMonth());
//Anio en que estamos
console.log(fecha.getFullYear());
//Hora, minutos, segundos y milisegundos
console.log(fecha.getHours());
console.log(fecha.getMinutes());
console.log(fecha.getSeconds());
console.log(fecha.getMilliseconds());

//Solo parte de la fecha
console.log(fecha.toString());
console.log(fecha.toLocaleString());
//Solo fecha
console.log(fecha.toLocaleDateString());
//solo hora
console.log(fecha.toLocaleTimeString());

//Para horas de retraso, comparar con otros paises, en guate es GMT-0600
console.log(fecha.getTimezoneOffset());
console.log(fecha.getUTCDate());
console.log(fecha.getUTCHours());

//cuantos segundos han pasado
console.log(Date.now());

//Fecha especifica
let miCumple = new Date(1995,6,1);
console.log(miCumple);
```

## Objeto Math

```
console.log(Math);
console.log(Math.PI);
console.log(Math.abs(-7.8)); //valor absoluto
//redondear
console.log(Math.ceil(7.8));
//Redondear al mayor
console.log(Math.floor(7.2));
console.log(Math.floor(7.8));
//redondear al mas proximo
console.log(Math.round(7.2));
console.log(Math.round(7.8));
//raiz cuadrada
console.log(Math.sqrt(8));
//potencia
console.log(Math.pow(2,5));
//indicador si es 0, positivo o negativo, retorna 0, 1, -1
console.log(Math.sign(0));
console.log(Math.sign(9));
console.log(Math.sign(-9));
//random entre 0 y 1
console.log(Math.random());
console.log(Math.round(Math.random() * 1000)); // 0 - 1000
```

## Operador de cortocircuito

```
/*
OR: cuando el valor de la izquierda en la expresion siempre
    pueda validar a true, es el valor que se cargara por defecto.

AND: cuando el valor de la izquierda en la expresion pueda
    validar a false, es el valor que se cargara por defecto */

function saludar(nombre){
    nombre = nombre || "Desconocido";
    console.log(`Hola ${nombre}`);
}

saludar("Gudiel");
saludar();

console.log(10 || "valor de la derecha");
console.log(null || "valor de la derecha");

console.log(10 && "valor de la derecha");
console.log("null" && "valor de la derecha");
```

## 30 - Alert, Confirm y Prompt

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <script>
    //window.alert("Esto es una alerta");
    //window.confirm("Esto es una confirmacion");
    //window.prompt("Esto es un aviso, permite al usuario ingresar valor");

    let alerta = window.alert("Esto es una alerta"); //undefined
    let confirmacion = window.confirm("Esto es una confirmacion"); //true o false
    let aviso = window.prompt("Esto es un prompt, permite al usuario ingresar valor");
    //null o lo ingresado
  </script>
</body>
</html>
```

## 31 - Expresiones Regulares

```
/*
Son una secuencia de caracteres que forma un patron de busqueda,
principalmente utilizada para la busqueda de cadenas de caracteres */

let cadena = "Lorem ipsum es el texto que se usa habitualmente en diseño gráfico en\
demostraciones de tipografías o de borradores de diseño para probar el diseño \
visual antes de insertar lorem el texto final.";

let expReg = new RegExp("lorem","i");//bandera g, significa global (todas las coincidencias)
                                     //bandera i, no diferencia entre mayusculas y minusculas
                                     //se pueden poner banderas juntas -> "ig"

let expReg2 = /lorem/ig;

console.log(expReg.test(cadena)); //probando que la expresion existe dentro del parrafo
console.log(expReg.exec(cadena)); //retorna un arreglo

console.log(expReg2.test(cadena));
console.log(expReg2.exec(cadena));
```

## 32 - Funciones anonimas autoejecutables

```
//VERSION CLASICA
(function(){
    console.log("Hola mundo CLASICA");
})();

//pasando parametros
(function(c){
    console.log("Hola mundo 2");
    c.log("Gudiel");
})(console);

//LA CROCKFORD
((function(){
    console.log("Hola mundo CROCKFORD");
}))();

//UNARIA
+function(){
    console.log("Hola mundo UNARIA");
})();

//FACEBOOK
!function(){
    console.log("Hola mundo FACEBOOK");
})();
```

## 33 – Modulos (import, export)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Modulos en JavaScript</title>
</head>
<body>
    <h1>Modulos en JavaScript</h1>

    <script src="js/modulos.js" type="module"></script>
    <script src="js/no-modulos.js" nomodule></script>
</body>
</html>

<!-- el 'nomodule' hace que no se duplique el codigo
    uno lo carga en ES6 y el otro solo en ES5'-->
```



no-modulos.js

```
console.log("No soporta en navegador +ES6");
```

modulos.js

```
import saludar, {PI,usuario} from "./constantes.js";
import {aritmetica} from "./aritmetica.js";

console.log("Archivo modulos.js");
console.log(PI, usuario);

console.log(aritmetica.sumar(2,3));

saludar();
```

constantes.js

```
export const PI = Math.PI;
export let usuario = "json";

//forma de exportar un default en variables
let password = "gudiel";
//export default password;

//Solo se puede exportar un default
export default function saludar(){
  console.log("Hola mudulos +ES6")
}
```

aritmetica.js

```
function sumar(a,b){
  return a+b;
}

function restar(a,b){
  return a-b;
}

export const aritmetica = {
  sumar,
  restar
};
```

## 44 – Temporizadores (setTimeout, setInterval)

```
//se ejecuta una vez despues de 3 segundos
let temp = setTimeout(()=>{
    console.log("Hola Mundo!",new Date().toLocaleTimeString());
}, 3000);

//se ejecuta siempre cada 10 segundos
let temp2 = setInterval(()=>{
    console.log("Hola Mundo 2!", new Date().toLocaleTimeString());
}, 10000);

//Deja de ejecutarse el setTimeout
clearTimeout(temp);
//Deja de ejecutarse el setInterval
clearInterval(temp2);
```

## 45 – Asincronia y el Event Loop

```
/*Codigo sincrono bloqueante*/
(()=>{
    console.log("Codigo sincrono");
    console.log("Inicio");

    function dos(){
        console.log("Dos");
    }

    function uno(){
        console.log("Uno");
        dos();
        console.log("Tres");
    }

    uno();
    console.log("Fin");
})();

console.log("*****");

/*Codigo Asincrono No bloqueante */
(()=>{
    console.log("Codigo Asincrono");
    console.log("Inicio");

    function dos(){
        setTimeout(function() {
            console.log("Dos");
        }, 1000);
    }
})
```

```

function uno(){
  setTimeout(function() {
    console.log("Uno");
  }, 0);
  dos();
  console.log("Tres");
}

uno();
console.log("Fin");
})();

```

## 46 - Callbacks

```

function cuadradoCallback(value, callback){
  setTimeout(() => {
    callback(value, value*value);
  }, 0 | Math.random() * 1000);
}

/*Se van ejecutando en orden*/
cuadradoCallback(0, (value,result) => {
  console.log("Inicia Callback");
  console.log(`Callback: ${value}, ${result}`);
  cuadradoCallback(1, (value,result) => {
    console.log(`Callback: ${value}, ${result}`);
    cuadradoCallback(2, (value,result) => {
      console.log(`Callback: ${value}, ${result}`);
      cuadradoCallback(3, (value,result) => {
        console.log(`Callback: ${value}, ${result}`);
        cuadradoCallback(4, (value,result) => {
          console.log(`Callback: ${value}, ${result}`);
          cuadradoCallback(5, (value,result) => {
            console.log(`Callback: ${value}, ${result}`);
          });
        });
      });
    });
  });
});

```

## 47 - Promesas

```
function cuadradoPromise(value){

  if(typeof value !== "number"){
    return Promise.reject(`Error, el valor " ${value} " ingresado no es numero`);
  }

  return new Promise((resolve, reject)=>{
    setTimeout(() => {
      resolve({
        value: value,
        result: value * value
      });
    }, 0 | Math.random() * 1000);
  });
}

cuadradoPromise(0)
  .then((obj)=>{
    console.log("Inicio Promise")
    console.log(`Promise: ${obj.value}, ${obj.result}`);
    return cuadradoPromise(1);
  })
  .then(obj=>{
    console.log(`Promise: ${obj.value}, ${obj.result}`);
    return cuadradoPromise(2);
  })
  .then(obj=>{
    console.log(`Promise: ${obj.value}, ${obj.result}`);
    return cuadradoPromise(3);
  })
  .then(obj=>{
    console.log(`Promise: ${obj.value}, ${obj.result}`);
    return cuadradoPromise(4);
  })
  .then(obj=>{
    console.log(`Promise: ${obj.value}, ${obj.result}`);
  })
  .catch(err => console.log(err));
```

## 48 – Async - Await

```
function cuadradoPromise(value){

    if(typeof value !== "number"){
        return Promise.reject(`Error, el valor " ${value} " ingresado no es numero`);
    }

    return new Promise((resolve, reject)=>{
        setTimeout(() => {
            resolve({
                value: value,
                result: value * value
            });
        }, 0 | Math.random() * 1000);
    });
}

async function funcionAsincronaDeclarada(){
    try {
        console.log("Inicio Async Function");
        let obj = await cuadradoPromise(0);
        console.log(`Async Function: ${obj.value}, ${obj.result}`);

        obj = await cuadradoPromise(1);
        console.log(`Async Function: ${obj.value}, ${obj.result}`);

        obj = await cuadradoPromise(2);
        console.log(`Async Function: ${obj.value}, ${obj.result}`);

        obj = await cuadradoPromise(3);
        console.log(`Async Function: ${obj.value}, ${obj.result}`);

    } catch (error) {
        console.log(error);
    }
}

funcionAsincronaDeclarada();
```

## 49 – Symbols

```
/*Son como variables privadas, que no muestra el nombre de la misma*/
const NOMBRE = Symbol();
const SALUDAR = Symbol();

const persona = {
  [NOMBRE]: "Gudiel"
}

console.log(persona);

persona.NOMBRE = "Christopher";
console.log(persona);
console.log(persona.NOMBRE);
console.log(persona[NOMBRE]);

persona[SALUDAR] = function(){
  console.log("Hola Mundo!");
}

persona[SALUDAR]();

console.log("-----");
for (const propiedad in persona) {
  console.log(propiedad);
  console.log(persona[propiedad]);
}

console.log(Object.getOwnPropertySymbols(persona));
```

## 50 - Sets

```
//El set elimina duplicados

const set = new Set([1,2,3,4,5,true,false,false,{},{},"hola","H0la"]);
console.log(set);
console.log(set.size);

const set2 = new Set();
set2.add(1);
set2.add(2);
set2.add(1);
console.log(set2);

for (const item of set) {
  console.log(item);
}
```

```

//se convierte el set a un arreglo
let arr =Array.from(set);
console.log(arr[0]);

//eliminando
set.delete("H01a");
console.log(set);

//verificando si existe
console.log(set.has("hola"));
console.log(set.has(19));

//limpiando
set2.clear();
console.log(set2);

```

## 51 – Maps

```

/*Son diccionarios, tienen una llave y un valor*/
let mapa = new Map();
mapa.set("nombre","Christopher");
mapa.set("apellido","Gudiel");
mapa.set("edad",35);

console.log(mapa);
console.log(mapa.size);
console.log(mapa.has("correo"));
console.log(mapa.has("nombre"));
console.log(mapa.get("nombre"));
mapa.set("nombre","Alexander");
console.log(mapa.get("nombre"));
mapa.delete("apellido");
console.log(mapa);

for (let [key,value] of mapa) {
    console.log(`Llave: ${key}, Valor: ${value}`);
}

const mapa2 = new Map([
    ["nombre","Gudiel"],
    ["Edad",10]
]);

console.log(mapa2);

//guardando en un array
const llavesMapa2 = [...mapa2.keys()];
const valoresMapa2 = [...mapa2.values()];

console.log(llavesMapa2);
console.log(valoresMapa2);

```

## 52 – WeakSets & WeakMaps

*/\*WeakMap es una colección similar a Map que permite solo objetos como propiedades y los elimina junto con el valor asociado una vez que se vuelven inaccesibles por otros medios.*

*WeakSet es una colección tipo Set que almacena solo objetos y los elimina una vez que se vuelven inaccesibles por otros medios \*/*

```
const ws = new WeakSet();

//solo aceptan objetos
let valor1 = {"valor1":1};
let valor2 = {"valor1":2};
let valor3 = {"valor1":3};
ws.add(valor1);
ws.add(valor2);
console.log(ws);

console.log(ws.has(valor1));
console.log(ws.has(valor3));

ws.delete(valor2);
console.log(ws);

ws.add(valor2);
ws.add(valor3);
console.log(ws);

// setInterval(() => {
//     console.log(ws)
// }, 1000);

const wm = new WeakMap();
let llave1 = {};
let llave2 = {};
let llave3 = {};

wm.set(llave1,1);
wm.set(llave2,2);
console.log(wm);

console.log(wm.has(llave1));
console.log(wm.has(llave2));

console.log(wm.get(llave2));

wm.delete(llave3);

console.log(wm);
```



## 53 – Iterables & Iterators

```
const iterable = [1,2,3,4,5];
//const iterable = "Hola mundo"; //pueden ser y tambien maps

//accedemos al iterador del iterable
const iterador = iterable[Symbol.iterator]();

console.log(iterable);
console.log(iterador);
//console.log(iterador.next());

let next = iterador.next();

while(!next.done){
  console.log(next.value);
  next = iterador.next();
}
```

## 54 – Generators

```
/*Van ejecutando codigo en cada yield*/
function* iterable(){
  yield "Hola";
  console.log("Hola consola");
  yield "Hola 2";
  console.log("Hola consola 2");
  yield "Hola 3";
  yield "Hola 4";
}

let iterador = iterable();
// console.log(iterador.next());
// console.log(iterador.next());
// console.log(iterador.next());
// console.log(iterador.next());

for (let y of iterador) {
  console.log(y);
}

//convirtiendo en array
const arr = [...iterable()];
console.log(arr);

function cuadrado(valor){
  setTimeout(() => {
    return console.log({
      valor,
      resultado: valor*valor
    });
  });
}
```

```

    });
    }, Math.random() * 1000);
}

function* generador(){
  console.log("Inicia Generator");
  yield cuadraro(0);
  yield cuadraro(1);
  yield cuadraro(2);
  yield cuadraro(3);
  console.log("Termina Generador")
}

let gen = generador();

for (let y of gen) {
  console.log(y);
}

```

## 55 – Proxies

```

//se usan para crear copias de objetos
const persona = {
  nombre:"",
  apellido:"",
  edad:0
}

const manejador = {
  set(obj,prop,valor){
    if(Object.keys(obj).indexOf(prop)===-1){
      return console.error(`La propiedad ${prop} no existe en el objeto persona`);
    }
    obj[prop] = valor;
  }
}

const gudiel = new Proxy(persona,manejador);
gudiel.nombre = "Gudiel";
gudiel.apellido = "Chris";
gudiel.edad = 26;
gudiel.twitter = "@gudiel"
console.log(gudiel);
console.log(persona);

```

## 56 – Propiedades dinámicas de los objetos

```
let aleatorio = Math.round(Math.random()*100 +5);
const objUsuarios = {
  [`id_${aleatorio}`]: "Valor aleatorio"
};
const usuarios = ["gudiel", "alexander", "chris", "acajabon"];

usuarios.forEach((usuario, index) => objUsuarios[`id_${usuario}`] = usuario);

console.log(objUsuarios);
```

## 57 – this

```
this.nombre = "Global";

function imprimir(){
  console.log(this.nombre);
}

imprimir();

const obj = {
  nombre: "Global 2",
  imprimir: function (){
    console.log(this.nombre);
  }
}

const obj2 = {
  nombre: "Global 3",
  imprimir: () => { //las funciones flecha mantienen la referencia (no la del objeto)
    console.log(this.nombre); //el this no hace referencia al nombre del objeto
  }
}

obj.imprimir();
obj2.imprimir();
```

## 58 – Call, apply, bind

```
//tendria que estar en un html para poder visualizar
this.lugar = "Contexto Global";

function saludar(saludo,aQuien) {
    console.log(`${saludo} ${aQuien} desde el ${this.lugar}`);
}

saludar();

const obj = {
    lugar: "Contexto objeto"
}

//saludar el global, y recibe un nuevo contexto 'obj'
saludar.call(obj, "hola", "Gudiel");
saludar.apply(obj, ["hola", "Gudiel"]);

const persona = {
    nombre: "Gudiel",
    saludar: function () {
        console.log(`hola ${this.nombre}`)
    }
}

persona.saludar();

const otraPersona = {
    saludar: persona.saludar.bind(persona) //bind enlaza el contexto de 'persona'
}

otraPersona.saludar();
```

## 59 – JSON (JavaScript Object Notation)

**JavaScript Object Notation** o Notación de Objetos de *JavaScript*, es un formato ligero de intercambio de datos. Leerlo y escribirlo es simple para humanos, mientras que para las máquinas es fácil de interpretar y generar. Todos los lenguajes de programación lo soportan.

### Sintaxis derivada de los objetos JavaScript:

- Los datos se encuentran en pares de nombre / valor.
- Los datos están separados por comas.
- Las llaves {} contienen objetos.
- Los corchetes [] contienen arreglos.
- Los datos tienen un nombre y un valor.
- Los datos se escriben como pares de nombre / valor "nombre" : "valor".

### Valores JSON:

- Un número (entero o de coma flotante).
- Una cadena (entre comillas dobles).
- Un booleano (verdadero o falso).
- Un arreglo (entre corchetes).
- Un objeto (entre llaves).
- Un valor nulo (null).

### Métodos JavaScript para JSON:

***JSON.parse()***: Analiza una notación *JSON* y la convierte en un tipo de dato *JS*.

```
console.log("*** JSON.parse ***");
console.log(JSON.parse("{}"));
console.log(JSON.parse("[1,2,3]"));
console.log(JSON.parse("true"));
console.log(JSON.parse("false"));
console.log(JSON.parse("19"));
console.log(JSON.parse(' "Hola Mundo" '));
console.log(JSON.parse("null"));
//console.log(JSON.parse("undefined"));
console.log(JSON.parse('{ "x": 2, "y": 3 }'));
```

***JSON.stringify()***: Convierte un tipo de dato *JS* a notación *JSON*.

```
console.log("*** JSON.stringify ***");
console.log(JSON.stringify({}));
console.log(JSON.stringify([1, 2, 3]));
console.log(JSON.stringify(true));
console.log(JSON.stringify(false));
console.log(JSON.stringify(19));
console.log(JSON.stringify("Hola Mundo"));
console.log(JSON.stringify(null));
console.log(JSON.stringify(undefined));
console.log(JSON.stringify({ x: 2, y: 3 }));
```

Aprendejavascript.org

Developer.mozilla.org

### Extensiones:

Live Server: Para ir viendo los resultados en tiempo real, abre el documento en 127.0.0.1:5500/...

- Se le da click derecho en cualquier parte y sale la opción: **Open with live server**
- Frenarlo en: **Stop live server**

Codepen.io: para probar el código online

Para probar código síncrono y asíncrono: <http://latentflip.com/loupe/>