# JAVASCRIPT GENERAL

## 1.1 UBICACIÓN JAVASCRIP HTML

**INTERNAMENTE EN <SCRIPT></SCRIPT>**

Dentro de etiquetas head o body

<script>

document.getElementById("saludo").innerText ="Hola Mundo";

</script>

**HOJA EXTERNA TERMINACIÓN .JSS**

Dentro de head o body.

<script src="app.js"></script>

<button type="button" onclick="clickeoElBoton()">Botón :D</button>

**VARIOS ARCHIVOS EXTERNOS:**

<script src="myScript1.js"></script>

<script src="myScript2.js"></script>

**REFERENCIAS EXTERNAS Y PATH:**

<script src="https://www.w3schools.com/js/myScript.js"></script>

<script src="/js/myScript.js"></script>

## 1.2 SALIDAS DE TEXTO

**innerHTML**

Define contenido HTML del tag.

<script>

  document.getElementById("demo").innerHTML = 5 + 6;

</script>

**document.write()**

Escribir en doc html. Ya escrito (onlick, etc final) borra todo html doc.

<script>

  document.write(5 + 6);

</script>

**window.alert() || alert()**

Alerta en ventana de nav.

<script>

  window.alert(5 + 6);

</script>

**console.log()**

Imprimir en consola.

console.log(5 + 6);

**window.print()**

Imprimir página

<button onclick="window.print()">Print this page</button>

## 1.3 DECLARACIONES

Programa == conjunto de declaraciones, exe en orden. Reglas:

**- PUNTO Y COMA AL FINAL.**

**- SALTOS DE LÍNEA DESPUÉS DE “=”.**

**- ESPACIOS Y ENTERS NO TOMADOS EN CUENTA.**

**- BLOQUES DE CÓDIGO DE DECLARACIONES O ENCERRADO.**

**KEYWORDS** palabras reservadas:

Var, let, const, if, switch, for, function, return, try, this.

## 1.4 SYNTAX

**LITERALS** números y strings (10, 11.4, “hola”).

**VARIABLES** almacenan datos.

let x = 6;

**OPERADORES** exe operaciones ej: +, -, \*, /, ++, --, +, -, !, !!

**EXPRESIONES** operaciones

5 \* 10

x \* 10

"John" + " " + "Doe"

**COMENTARIOS** En línea //así multilínea /\*así\*/

**IDENTIFIERS** reglas:

- Comienza con letra (A-Z, a-z), signo de dólar ($) o (\_)

- Son Case sensitive, Camel case == primer palab. Minus, otras empiezan mayus.

- Se puede guiones y guiones bajos.

**UNICODE** click [aquí](https://www.w3schools.com/charsets/ref_html_utf8.asp) para ver unicodes.

## 1.5 OPERADORES

**OPERADORES ARITMÉTICOS**

**+ ** suma -  resta \*  multiplicación \*\*potencia / división

%  residuo de división ++  incrementar --  disminuir

**OPERADORES DE ASIGNACIÓN**

=  asigna vals. +=  suma y asigna -=  resta y asigna \*=  multiplica y asigna

/=  divide y asigna %=  módulo y asigna \*\*= potencia y asigna

**OPERADORES LÓGICOS**

&&  y ||  o !  no

**OPERADORES DE TYPO**

**typeof** identifier || literal  el tipo de una variable o literal

objeto **instanceof** constructor  si objeto viene de un constructor

## 1.6 TIPOS DE DATOS

**1) String ** let x = “string”, let x = ‘string’

**2) Number ** let x1 = 3; let x2 = 34.8;

Notación exponencial: let x1 = 123e5; // 12300000 let x1 = 123e-5; //0.00123

**3) BigInt ** números muy grandes ej: let x = BigInt(“123456789012345678901234567890”) // num+n

\*NOTA operar solo con num+n: x+ 2n; x-3n; x/2n; x\*\*2n,

\*NOTA 2:: usar bigint para no notación exponencial

**4) boolean ** true / false == con testeos, condicionales

**5) array = let x = [1, 2, 3];**

**6) Object = const person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};**

**7) Undefined =** sin valor let x; .

## 1.7 FUNCIONES

**1.7.1 SYNTAX**

function name(parameter1, parameter2, parameter3) {

// code to be executed }j

**1.7.2 INVOCACIÓN**

identifier(param1, param2...);

**1.7.3 RETURN**

return → función == valor. Se puede asignar y operar como valor.

function toCelsius(fahrenheit) {  
  return (5/9) \* (fahrenheit-32);  
}

**1.7.4 VARIABLES LOCALES**

Declaradas dentro → scope solo dentro función.

// code here can NOT use carName  
function myFunction() {  
  let carName = "Volvo";  
  // code here CAN use carName  
}  
// code here can NOT use carName

## 1.8 OBJETOS

**1.8.1 SYNTAX** pares == {propiedad: valor, propiedad2: valor2...}

const person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};

**1.8.2 ACCESO**

1) *objectName.propertyName*

*2) objectName["propertyName"]*

**1.8.3 MÉTODOS**

funciones como valor de propiedades.

const person = {  
  firstName: "John",  
  lastName : "Doe",  
  id       : 5566,  
  fullName : function() {  
    return this.firstName + " " + this.lastName;  
  }  
};

**1.8.4 THIS KEYWORD**

Casos:

1) Refiere al objeto propio.

2) (Solo) refiere al global object

3) (función) refiere a objeto q pertenece (glob. O método)

4) (Strict mode) this == undefined

5) (Event) this == refiere al elemento que recibe evento.

6) call(), apply(), bind(). This == cualquier objeto.

**1.8.5 ACCEDER OBJETOS**

Syntax:

let variable = objectName.methodName(param1, param2, param\_n);

## 1.9 EVENTOS

Acciones sobre elementos HTML.

**1.9.1 SYNTAXIS**

<elementevent='some JavaScript'>

<element event="some JavaScript">

Ejemplo:

<button onclick="document.getElementById('demo').innerHTML = Date()">The time is?</button>

<button onclick="this.innerHTML = Date()">The time is?</button>

**1.9.2 EVENTOS COMUNES**

1) onchange → cambio elemento html

2) onclick → click en elemento

3) onmouseover → hover elemento

4) onmouseout → mouse alejar elemento

5) onkeydown → presiona letra teclado

6) onload → nav. Termina cargar pg.

## 1.10 STRINGS

Líneas texto (0-más ch)

**1.10.1 LENGTH**

string.length; → dígitos string.

**1.10.2 ESCAPES**

Colocar símbolos propios JS y mod strings

1) \’ == ‘

2) \” == “

3) \\ == \

4) \ → continuar str sig linea (aplica en .innerHTML y .innerText)

let a = "Juan \

José";

5) \n → nueva línea ej: let a = "Juan \nJosé";

6) \t → tab horizontal ej: let a = "Juan\tJosé";

7) \v → tab vertical ej: let a = "Juan\vJosé";

## 1.11 MÉTODOS STRING \* no remplaza

**1.11.1 EXTRAER PARTES \* no remplaza**

1) **slice(idx\_inicio, inx\_final);**  | no incluye idx final.

**slice(idx\_inicio);**  → inicio hasta final

**slice(-idx, -idx); →** -idx == referencia de derecha-izquierda.

**2) substring (idx\_inicio, idx\_final); | ==** slice → toma index inicio en 1

**3) substr(indx\_inico, length); |** == slice, 2do. Param pide tamaño

**substr(indx\_inicio);**  | toma del index → final

**substr(-indx\_inicio, -indx\_final);**  == cuenta atrás → adelante

**1.11.2 REMPLAZAR PARTES STRING**

**1) replace(“plbr”, “remplazo”);** == No cambia, retorn. Nuevo str. (solo primer, case sensitive)

**replace(/palbr/i, “remplazo”); ==** sin case sensitive

**replace(/palbr/g, “remplazo”); ==** rplz. Todo match.

**2) replaceAll(); ==** replace | permite regexp (/galsdj/g siempre).

**1.11.3 MAYÚSCULAS Y MINÚSCULAS, CONCATENAR**

**1) string.toUpperCase();** → mayúsculas

**2) string.toLowerCase(); →** minúsculas

**3) str.concat(txt1, txt2, etc); →** concatena texto

**1.11.4 ELIMINAR ESPACIO**

**1) strng.trim();**  → remueve ambos lados.

**2) str.trimStart();**  → == trim, solo remv inicio

**3) str.trimEnd(); → ==** trim, remv final

**1.11.5 AÑADIR DÍGITOS INICIO-FINAL**

**1) str.padStart(length, “digito”);**  dígito izquierda → str.length == **length**

**2) str.padEnd(length, “dígito”);**

**1.11.6 EXTRAER CARÁCTERES**

**1) charAt(index); →** extrae carácter || index == no existe == string vacía

**2) charCodeAt(index); →** carácter → unicode utf-16

**3) string[index];** → mismo charAt || index == no exist == undefined

**1.11.7 STRING → ARREGLO**

**string.split(“carácter”);**  | carácter == separación == no incluye

**1.11.8 BÚSQUEDA EN STRING**

**1) string.indexOf(“word”);** == index primera letra| no hay == -1

**2) string.lastIndexOf(“word”, pos\_inicial (opt)); ==** index último match | nohay == -1

**3) string.search(“string”||regexp);** → index inicial

**4) string.match(“word”); →** arreglo con todos matches || regexp

**5) string.matchAll(“word”);**  iterador === [...string.matchAll(“word”)]

**6) string.includes(“word”);**  si == true, no == false

**7) string.startsWith(“word”);**  → true || false (case sensitive)

**8) string.endsWith(“word”);** → true || false (case sensitive)

## 1.12 STRINGS LITERALES

**let** text = `van así dijo “monchito” y ‘juanito’`; → multilínea

**1.12.1 INTERPOLACIÓN**

**let** price = 10;

**let** vat = 0.25;

**let** total = `Total: ${(price\*(1+vat)).toFixed(2))}`;

## 1.13 NÚMEROS

**let x** = 3; **let y** = 3.14; **let z =** 123e5; (12300000); **let k** = 123e-5; (0.00123)

**Rango seguro**: 9007199254740991 y -9007199254740991.

**1.13.1 APROXIMACIÓN**

Después de 15 dígitos.

**1.13.2 OPERADORES**

**+ →** Concatena strings y suma números (preferencia strings)

**-, /, \*, \*\*, % →** Opera, si ambos Strings == convierte números

**Number(valor) ==** convierte número

**1.13.3 NaN – Not a Number**

→ Aritmética número y no número

**1) isNaN(número);**  → true == NaN, false != NaN

**2) INFINITO (**Infinity o -Infinity)

→ Número fuera d rango, x/0.

## 1.14 BIGINT

**Syntax:**

**1)** let x = 9999999999999n;

**2)** let y = BigInt(123492834928374);

typeof x == bigint

→ Decimales == no.

**1) Number.isInteger(**número**);**  → true == entero, double == false

**2) Number.isSafeInteger(**número**);**  → true == dentro rango

## 1.15 MÉTODOS NÚMEROS

**1.15.1 CONSTANTES**

Math.E //return uler's number  
Math.PI// returns PI  
Math.SQRT2// returns the square root of 2  
Math.SQRT1\_2// returns the square root of 1/2  
Math.LN2// returns the natural logarithm of 2  
Math.LN10// returns the natural logarithm of 10  
Math.LOG2E// returns base 2 logarithm of E  
Math.LOG10E// returns base 10 logarithm of E

**1.15.2 GENERALES**

**1) literal/variable/expresión.toString();** → retorna número en str

**2) Number(“NÚMERO”);** cvrt string, boolean, etc → número

**3) parseInt(**“número”**);**  → string to número, número hasta punto, espacio, no número. 1Dgt= num.

**1.15.3 OBJETO “NUMBER”**

**1) Number.isInteger(**literal/variable/expresión**);**  → true == entero

**2) Number.isSafeInteger(**literal/varialbe/expresión**); →** true == en rango.

**3) Number.parseFloat(“**string”**);**  → número (tmbn decimales) hasta espacio, letra.

**4) Number.parseInt(“**string”**);**  → str a numero hasta punto, espacio, letra.

**1.15.4 OBJETO MATH**

**1.15.4.1 números a enteros**

1) Math.round(x); 🡪 int + cerca

2) Math.ceil(x); 🡪 aprox hacia arriba

3) Math.floor(x); 🡪 aprox abajo

4) Math.trunc(x); 🡪 part int de “x”

**1.15.4.2 operaciones**