**JS**

Forma de enlazar JS (recomendada)

Dentro del archivo HTML, al final del body...

**<script src=”codigo.js”> </script>**

**Variables**

*Espacio almacenado en memoria*

Tipos de datos (primitivos):

- String (cadena de texto)

variable = “cadena de texto”;

- Number

variable = (nro);

- Boolean

variable = true/false;

- Symbol

(variables no definidas / error)

- Undefined

Variable existente pero no inicializada

- Null

Valor nulo para asignar a variables

- Nan

Not a number (no es un numero - error de operaciones matematicas)

// scope = bloque de alcance de una variable

// hosting = flujo de lectura en programacion

**Formas de declarar variables**

- **var** / variable declarada con un alcance global

**Sintaxis: var nombre\_variable = valor;**

- **let** / variable declarada con un alcance de su respectivo bloque

**Sintaxis: let nombre\_variable = valor;**

- **const** / variables declaradas con un valor constante (no cambian)

**Sintaxis: const nombre\_variable = valor;**

**Prompt()**

Funcion para almacenar valores ingresados x el usuario

**Sintaxis: let nombre\_variable = prompt(“Ingrese valor: ”);**

**Operadores en JS**

**Operadores Aritmeticos**

Toman valores numericos (ya sean literales o variables) como sus operandos y retornan un valor numerico único.

variable-- (decremento)  
variable++ (incremento)

**Operadores de asignacion**

Asigna un valor al operando de la izquierda basado en el valor del operando de la derecha

**x = y**

**x += y ( x = x + y)** \*\* -> exponencial

**Concatenacion**

Unir strings

Convierte cualquier valor en un string

*Forma de concatenar*

“*hola” + “ como estas” = “hola como estas”*

Metodo: .concat(valor / variable) -> concatena los valores del parametro

Frase = `mi nombre es ${nombre\_variable} y estoy caminando`;

(uso de backtick -> alt + 96 == `)

Escape de comillas -> uso de comillas dentro de strings (solucion, iniciar string con otro tipo de comilla)

**Operadores**

**Operadores Logicos**

Los operadores logicos nos devuelven un resultado a partir de que se cumpla (o no) una condicion, su resultado es booleano, y sus operandos son valores logicos

**&& (AND)**

**ll (OR)**

**! (NOT)**

**Operadores de Comparacion**

Los operadores de comparacion comparan dos expresiones y devuelven un valor Boolean que representa la relacion de sus valores (true / false)

**== (igualdad (no diferencia el tipo de dato))**

**!= (desigualdad (no diferencia el tipo de dato))**

**=== (identicos (extricto en el tipo de datos))**

**!== (no identicos (extrico en el tipo de datos))**

**>**

**>=**

**<**

**<=**

**camelCase**

JS trabaja con la metodologia camelCase

**Condicionales**

**If**

**else if**

**else**

**parseInt(variable)**

Funcion para convertir una variable en tipo de dato entero

**Arrays - Arreglos (objetos)**

**Sintaxis:   
 let nombre\_array = [“string”, 24, “string”];  
 let frutas = [“banana”, “pera”, “manzana”];**

Las posiciones de los elementos comienzan de 0 a X

***Mostrar elementos de los arrays***

Para mostrar un unico elemento debemos llamar al array en la posicion a mostrar

**Sintaxis:**

**nombre\_array[nro\_posicion];**

**frutas[0] == “banana” indice = posicion**

**Arrays Asociativos (objetos)**

Array con un valor asociado

**Sintaxis: let nombre\_array\_as = {**

**propiedad1: valor,**

**propiedad2: valor,**

**propiedad3: valor,**

**propiedad4: valor**

**};**

**Mostrar elementos de los arrays asociativos**

**Sintaxis: nombre\_array\_as[propiedad1];**

**Bucles e Iteraciones**

**- While** (mientras)

**Sintaxis:**

**while(condicion) {**

**Bloque de codigo**

**}**

**Do**

**Sintaxis:**

**do {**

**Bloque de codigo**

**}while(condicion);**

//break: detiene la ejecucion de un bucle

**For (declaramos, condicion, aumento-decremento)**

**Sintaxis:**

**for (let i = 0; i < x; i++) {**

**Bloque de codigo**

**}**

//continue: saltea lo siguiente del bloque del codigo

**Fot in** (para arreglos)

Devuelve la posicion de un objeto

**Sintaxis:**

**for (animal in animales) {**

**Bloque de codigo**

**}**

**Fot of** (para arreglos)

Devuelve el valor de un objeto

**Sintaxis:**

**for (animal of animales) {**

**Bloque de codigo**

**}**

//label: le asigna un nombre a un bucle

Sintaxis:

**nombreBucle:**

**for (animal of animales) {**

**Bloque de codigo**

**}**

**Funciones**

Sintaxis declaracion:

**function nombre\_funcion(parametros o no) {**

**Bloque de codigo**

**Return o no;**

**}**

**const nombre\_funcion = function(parametros o no) {**

**Bloque de codigo**

**Return o no;**

**}**

**Funciones flechas**

Sintaxis declaracion:

**const saludar = (parametros o n)=> {**

**Bloque de codigo**

**Return o no;**

**}**

Sintaxis llamarlas:

**nombre\_funcion(parametros o no);**

**Programacion Orientada a Objetos (POO)**

**Conceptos Basicos:**

- Clase

Plantilla creada para crear objetos (fabrica de objetos)

- Objetos

Son los resultados de las clases

- Atributos

Las particularidades de un objeto

- Metodos

Las cosas que pueden hacer nuestro objeto

- Constructor

Nos construye las propiedades del objeto

- Instanciacion

Finalizacion del objeto

Sintaxis general (ejemplo):

class animal {

constructor(especie, edad, color) {

this.especie = especie; (atributos)

this.edad = edad;

this.color = color;

}

(funciones dentro de una clase son metodos)

this.verInfo() {

Document.write(this.color);

}

}

const perro = new animal(“perro”, 4, “rojo”);

**Caracteristicas de la POO**

-Abstaccion

Singularizar metodos complejos en uno mas general

-Modularidad

Dividir un algoritmo en varios pedacitos para volver a ser utilizados

-Encapsulamiento  
Los datos son privados, encapsulados para no acceder de manera sencilla

-Polimorfismo

Consiste en ver como un objeto se comporta de manera distinta ante el mismo metodo, segun sus propiedades

**Otros conceptos de POO**

- Herencia

Para heredar se requiere de una nueva clase donde se le aplicaran los atributos(a eleccion) y metodos de otras clases

Sintaxis ejemplo anterior:

class Perro extends Animal {

constructor(especie, edad, color, raza) {

super(especie, edad, color);

this.raza = raza;

}

ladrar(){

alert(“¡WAW!”);

}

}

- Metodos estaticos

Sirve para llamar a un metodo de una clase sin necesidad de crear un objeto en si mismo

Sintaxis:

Static nombre\_metodo() {

Accion del metodo

}

- Metodos accesores ( Getters, Setters)

Setters

Sirve para modificar el valor de un atributo gracias a un metodo

Sintaxis:

set set\_nombre\_metodo(parametro) {

this.nombre\_atributo = parametro;

}

Nombre\_objeto.nombre\_metodo = valor;

Getters

Sirve para obtener informacion

Sintaxis:

get get\_nombre\_metodo() {

Return this.nombre\_atributo;

}

Nombre\_objeto.get\_nombre\_metodo;