JavaScript

Isomorfismo

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

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

Características:

* Lenguaje de alto nivel
* Interpretado
* Dinámico
* Débilmente tipado
* Multiparadigma
* 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 $ o
* Un guion bajo:
* Nunca con números o caracteres especiales

Tecnicas:

Usa ***snake\_case*** en:

* Archivos:
  + mi\_archivo\_javascript.js

Usa **UPPER\_CASE** en:

* Constantes:

Const UNA\_CONSTANTE = ‘Soy una constante’,

PI = 3.14159265

Usa UpperCamelCase en:

* Clases:

class SerHumano {

    constructor (hombre,genero){

        this.nombre = nombre

        this.genero = genero

    }

    miNombreEs(){

        return `Mi nombre es ${this.nombre}`

    }

}

Usa ***loweCamelCase*** en:

* **Objetos:**

const unObjeto = {

    nombre:'Jhonatan',

    email: 'jonmicha@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

1. **Primitivos:** Se accede directamente al valor.
   * String
   * Number
   * Boolean
   * Null
   * Undefined
   * NaN
2. **Compuestos:** Se accede a la referencia del valor.
   * Object = {}
   * Array = []
   * Function = () {}
   * Class {}
   * Etc.

El ámbito de las variables

Usar **var** hoy es una mala práctica por que se provoca **hoisting** una elevación de dichas variables, puesto que estas se vuelven globales y **let** genera variables de bloque

**const** se utiliza cuando queramos asignar un valor que no va a cambiar **y aplica solamente en valores primitivos mas no en compuestos.**

Cadenas de texto **String**

toUpperCase () ***Método para poner texto en MAYÚSCULAS***

toLowerCase () ***Método para poner texto en minúsculas***

Template Strings

Concatenar: es la acción de unir (textos, números)

   //Concatenación A N T E S

        let nombre = "Jose";

        let apellido = "Ramírez";

        let saludo = "Hola mi nombre es " + nombre + " " + apellido + ".";

        console.log(saludo)

        //Concatenación A H O R A

        //Interpolación de variables

        //Templates String

        let saludo2 = `Hola mi nombre es ${nombre} ${apellido}.`

        console.log(saludo2);

Números

toFixed(): ***Es un método para definir cuantos decimales queremos para un numero***

parseInt(): ***Método que pasa un texto a un numero entero***

parseFloat(): ***Método que pasa un texto a un numero con todo y su decimal***

typeof: ***Nos indica cual es la naturaleza de una variable***

Null, Undefined, NaN.

Undefined:  ***Indica una variable vacía, pero lo asigna directamente Js***

Null: ***Indica igualmente una variable vacía, pero en este caso es intencional por parte del programador.***

NaN: (Not a Number) ***Es un error cuando tratamos de hacer una operación aritmética entre valores que no sean valores numéricos.***

Valores Compuestos.

*Funciones*

Una función es un bloque de código, autocontenido, que se puede definir una vez y ejecutar en cualquier momento. Opcionalmente, Una función puede aceptar parámetros y devolver un valor.

Las funciones en JavaScript son objetos, un tipo especial de objetos:

Se dice que las funciones son ciudadanos de primera clase por que pueden asignarse a un valor, y pueden pasarse como argumentos y usarse como un valor de retorno.

        //Funcion declarada

        function estoEsUnaFuncion() {

            console.log ("Hola Mundo");

        }

            //Invocar una función

            estoEsUnaFuncion();

    //Asignar argumentos por defecto

        function saludar (nombre = "Anónimo", edad = 0){

            console.log(`Hola mi nombre es ${nombre} y tengo

            ${edad} años :D`)

        }

        saludar();

Debido al **Hoisting** que hay entre funciones y variables nosotros podemos declarar las funciones e invocarlas en cualquier parte del cuerpo de nuestro código y estas se van a elevar al ámbito global

  //Funciones declaradas VS funciones expresadas

        funcionDeclarada();

            function funcionDeclarada() {

                console.log(`Esto es una función declarada

                puede invocarse en cualquier parte de nuestro

                código, incluso antes de que la función

                sea declarada`);

            }

        functionDeclarada();

       //FUNCION E-X-P-R-E-S-A-D-A o Funcion ANONIMA

        //Se guarda dentro de una variable

        funcionExpresada();

        const funcionExpresada = function (){

            console.log(`Esto es una funcion expresada, es decir

            una funcion que se le ha asignado como valor

            a una variable, si invocamos esta funcion antes de

            su definicion JS nos dira...`)

        }

        funcionExpresada();

//LANZA UN ERROR PUESTO QUE NO SE APLICA EL HOISTING ENTONCES NO PODEMOS MANDAR A LLAMAR UNA FUNCION ANTES DE QUE SEA DELCARADA

Arreglos

  //N U E V A FORMA DE DEFINIR A R R E G L O S

        const c = Array.of("X","Y","Z",9,8,7);

        console.log(c);

  // GENERAR MULTIPLES ARREGLOS CON EL METODO .fill

        const d = Array(500).fill(true);

        console.log(d);

**Método .push**

    //AGREGAR UN NUEVO ELEMENTO A UN ARREGLO, AL FINAL

        const colores = Array.of("Azul", "Verde", "Rojo");

            console.log(colores)

        colores.push("Negro")

             console.log(colores)

**Metodo .pop**

   // ELIMINAR EL ULTIMO ELEMENTO \*METODO .pop\*

         const colores = Array.of("Azul", "Verde", "Rojo");

         colores.pop();

         console.log(colores)

**Metodo .forEach**

   // METODO .forEach

    // ES UN METODO DE LOS ARREGLOS QUE RECIBE UNA FUNCION

        colores.forEach(function(el,index){

            console.log(`<li id="${index}">${el}</li>`)

        })

Objetos en JavaScript

Un objeto es una colección de características. y consta de un nombre clave y un tipo de dato

const b = {}

    console.log(b);

const jose ={

    nombre:"Alberto",

    apellido:"Ramirez",

    edad:21,

    pasatiempos:["Hacer Ejercicio","Series","Musica"],

    soltero:false,

    contacto:{

        email:"joe.broken.com",

        num:123,

        fb:"JoseRamirezOz"

    },

    saludar:function(){

        cosole.log(`Hola :)`)

    },

}

console.log(jose.nombre)

/\*INFORMACION IMPORTANTE!

    -> DENTRO DE UN OBJETO A LAS VARIABLES SE LE VAN A LLAMAR ATRIBUTOS

    Y A LAS FUNCIONES SE LE VAN A LLAMAR METODOS

\*/

This, es una forma de hacer referencia a si mismo

Tipos de operadores

/\*TIPOS DE OPERADORES ----------------------------------------------

-----------------------------------------------------------------

-------------------------------------------------------------------

--------------------------------------------------------\*/

//OPERADORES ARITMETICOS ( + - \* / %)

//OPERADORES RELACIONALES (<, >, >=, <=, = , ==, ===, !=, !==)

/\*

    = Es un operador de asignacion

    == Comparacion de valores

    === Comparacion de tipo de dato y de valor

\*/

/\*Incremento Decremento\*/

let i = 2;

// i = i+2;

// i = i+3:

//Operador Unitario

i++;

i--;

++i;

--i;

Condicionales:

/\*OPERADOR TERNARIO (CONDICION) ? VERDADERO:FALSO\*/

let edad = 21;

let  eresMayor = (edad>=18)?"Eres mayor de Edad":"EresMenor";

console.log(eresMayor)

//ESTRUCTURA SWITCH CASE

/\*

Domingo 0

Lunes 1

Martes 2

Miercoles 3

Jueve 4

Viernes 5

Sabado 6

\*/

let dia = 1;

switch(dia){

    case 0:

        console.log("Domingo");

    break;

    case 1:

        console.log("Lunes");

    break;

    case 2:

        console.log("Martes");

    break;

    case 3:

        console.log("Miercoles");

    break;

    case 4:

        console.log("Jueves");

    break;

    case 5:

        console.log("Viernes");

    break;

    case 6:

        console.log("Sabado");

    break;

    default:

        break;

}

Manejo de errores.

//PERSONALIZACION DE ERRORES

/\*LA FUNCION isNaN DEVUELVE VERDADERO O FALSO

EVALUA SI EL CARACTER INSERTADO ES NotANumber

    TRUE= CUANDO EL VALOR NO ES UN NUMERO

    FALSE= CUANDO EL VALOR ES UN NUMERO

\*/

try {

    let numero = "y"

    if (isNaN){

        throw **new** Error (`El caracter insertado no es

        un numero`)

    }

    console.log(numero\*numero)

} catch(error){

    console.log(`Se produjo el siguiente ${error}`)

}

BREAK AND CONTINUE

/\*BREAK Y CONTINUE

    BREAK = FUNCIONA PARA ROMPER LA EJECUCION

        Y SALIR DE LA MISMA

    CONTINUE= SALTA LA ITERACION Y CONTINUA CON

        LOS DEMAS ELEMENTOS

        \*/

const numeros = [1,2,3,4,5,6,7,8,9,0];

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

    if (i === 5){

        break;

    }

    console.log(numeros[i])

}

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

    if (i === 5){

        continue;

    }

    console.log(numeros[i])

}

Desestructuración:

/\*--------------------------------------------------------

Desestructuración:

---------------------------------------------------------\*/

const numeros = [1,2,3];

//SIN DESESTRUCTURACION

let uno = numeros[0],

    dos = numeros[1],

    tres = numeros[2];

console.log(uno,dos,tres)

//CON DESESTRUCTURACION

const [one,two,three] = numeros;

console.log(one,two,three);

let persona = {

    nombre:"Jose",

    apellido:"Ramirez",

    edad:21

}

let {nombre,apellido,edad} = persona;

console.log(nombre,apellido,edad)

OBJETOS LITERALES

/\*OBJETOS LITERALES\*/

//DECLARACION DE VARIABLES

let nombre = "Kenai",

    edad = 7;

// OBJETO LITERAL

const perro = {

    nombre: nombre,

    edad:edad,

    ladrar: function(){

        console.log("guauu guauu")

    }

}

//NUEVA FORMA DE DEFINIR EL OBJETO

const dog = {

    nombre,  /\*ASUME QUE SE TOMA EL VALOR,

    QUE HAY EN LA VARIABLE CON EL MISMO NOMBRE\*/

    edad,  /\*ASUME QUE SE TOMA EL VALOR,

    QUE HAY EN LA VARIABLE CON EL MISMO NOMBRE\*/

    raza: "Callejero",

    ladrar(){ //ELIMINACION DE LOS DOS PUNTOS:

        console.log("guauu guauu guauu!!!!")

    }

}

PARAMETROS REST & OPERADOR SPREEAD

/\*-----------------------------------------------------

PARAMETROS REST & OPERADOR SPREEAD ----------------------------

------------------------------------------------------\*/

/\*REST - SE UTILIZA CUANDO POR EJEMPLO EN UNA FUNCION

NO SABEMOS CON EXACTITUD CUANTOS PARAMETROS

LE VAMOS A PASAR \*/

function sumar (a,b, ...c){

    let resultado = a +b ;

    c.forEach(

        function(n){

            resultado += n

        }

    );

    return resultado

}

/\*OPERADOR DE PROPAGACION, SPREEAD OPERADOR\*/

const arr1 = [1,2,3,4,5],

      arr2 = [6,7,8,9,0];

console.log(arr1, arr2);

//spreed ...

const arr3 = [...arr1,...arr2];

console.log(arr3);

Prototipos

Son la referencia a **los objetos con sus (propiedades y métodos)** que nos permitirán de manera muy simple construir nuestro nuevo objeto con las características que nosotros queremos plantear