***APUNTES JAVASCRIPT***

***Arrays***

Un array es un contenedor con diferentes objetos, ordenados de una manera determinada.

Los objetos dentro del array están desde la posición 0 en adelante.

Contenedor = [Objeto0, Objeto1, Objeto2]

Para llamar a alguno de los objetos solo hay que seleccionar el contenedor y dentro de [] la posición del objeto a elegir. Tomando el ejemplo del contenedor Computadora.

Computadora = [“Amd”,“16GB”, “1TB”,]

*Document.*write(Computadora[0]) Esto daría = AMD

Al llamar al numero 0 llamaríamos al objeto AMD y así sucesivamente .

¿Pero qué pasaría si quisiéramos llamar al objeto por su nombre?

Por ejemplo, si quisiéramos saber que PROCESADOR tiene nuestra computadora.

Debemos nombrar a cada objeto con un nombre y dos puntos. Como en este ejemplo.

Computadora = {

Procesador: “Amd”,

Ram: “16GB”,

Almacenamiento: “1TB”,

} ;

*Document.*write(Computadora[“Procesador”]) ; Esto daría = AMD

Poniendo el nombre del objeto entre comillas podremos llamar a cada objeto por su nombre, sin necesidad de recordar la posición de cada uno.

***Bucles e Interacción***

***While***

Es un bucle que al igual que *If* hace una pregunta y si la condición da *true* completa una acción. En cambio *While* hace la misma pregunta en bucle hasta que la respuesta de *false*.

*Aca vemos la diferencia:*

Numero = 0 ;

*If* (numero < 10) { *while* (numero < 10){

numero++ ; numero++ ;

*Document.*write(numero) *Document.*write(numero)

} }

**1 12345678910**

En este caso el *if* pregunta una vez si el numero es menor a 10 y como se cumple pasa el **0** a **1.**

En cambio el *while* lo pregunta una vez, hace que se incremente a 1, pero como 1 sigue siendo menor a 10, lo vuelve a incrementar y pasa a ser 2. Y así sucesivamente hasta que el numero deje de ser menor a 10.

***Do While***

*do* a diferencia de *while*, ejecuta antes de preguntar. Como el nombre lo indica *do* significa hacer o ejecutar. La forma correcta de utilizarlo es colocar el *do* antes que el *while* y si el while no se cumple, corta el bucle, pero ya habiéndose ejecutado 1 sola vez en el *do*.

Por ejemplo:

Numero = 0 ;

*do* {

*Document.*write(numero);

numero++;

} ;

*If* (numero > 10) {

*Document.*write(numero);

numero++ ;

}

Resultado: 0 Ya que numero no es mayor a 10, solo se ejecuta lo que hay en el do, y se corta el bucle.

***Break***

El *Break* se utiliza para cortar un bucle cuando se cumpla una condición. Por ejemplo.

Numero = 0 ;

*while* (numero < 1000){ En este caso como numero es menor a 100

numero++ ; el programa se ejecutará 1000 veces.

*Document.*write(numero); }

Numero = 0;

*while* (numero < 1000) { En este caso el *if* indica que cuando el número

numero++ ; valga 10 se ejecutara *break* y detendrá el bucle.

*Document.*write(numero) ;

*If* (numero == 10) {

*break*; }

} ;

***For***

***For in***

Recorre un array, mostrando el índice de la posición de cada elemento en el array.

* Itera sobre las propiedades enumerables de un objeto.
* Recorre todas las propiedades enumerables, incluyendo aquellas heredadas de la cadena de prototipos.
* Se utiliza principalmente para objetos, no para iterar sobre elementos de datos en estructuras como arrays o cadenas.

***For of***

* Itera sobre elementos iterables, como arrays, cadenas, mapas, sets, etc.
* No accede a las propiedades del objeto, sino a los valores de los elementos.
* Proporciona una forma más simple y legible para iterar sobre elementos en comparación con **for...in**.

***Continue***

Hace lo mismo que el break pero en vez de cortar el bucle saltea la condición que hay en el if. Por ejemplo, si quisiéramos contar del 1 al 20 pero salteando el 10 ¿Cómo haríamos?...

Numero = 1;

*while* (numero < 1000) { En este caso el *if* indica que cuando el número

numero++ ; valga 10 se ejecutara *break* y detendrá el bucle.

*Document.*write(numero) ;

*If* (numero == 10) {

*break*; }

} ;

***Funciones***

Con las funciones podemos darle un nombre a un conjunto de códigos y ejecutarlo varias veces solo nombrando la función.

function nombre(){

    Variable = prompt ("Hola, como estuvo tu día?");

    if (variable == "bien") {

        alert("Me alegro")

    }

     else {

        alert ("Ah mira vo")

    }

}

***Return***

Si una función solo hace una acción pero no se queda con un valor, no tendrá ningún tipo de dato y va a quedar con indefinido

function nombre(){

    alert("Acción");

    return "Dato de la función"; {

}

El Return le dice a la variable que tipo de dato va a ser. También funciona como el break al ejecutarlo corta el flujo de la acción, es decir que si ponemos el Return antes de la acción, la acción no se accionara.

***Parámetros***

Las funciones que no pueden cambiar sus valores solo nos sirven para una acción en concreto y nada más, pero si le damos la opción que haga la misma acción con diferentes parámetros. Eso la vuelve mas útil. Por ejemplo, si llamamos a una variable Suma y queremos que sume dos números, no podemos definirle dos números ya que lo único que va a hacer será siempre la misma operación.

Por ejemplo, suma = 2+5; Esta función siempre sumara 2 + 5.

Para esto sirven los parámetros, para hacer la misma acción pero cambiando los valores de sus parámetros.

***Funciones flecha***

const saludar = (nombre)=>{  }

Otra forma de escribir funciones, y sirva para funciones cortas con una sola expresión, esta función se retorna sola a la vez que se ejecuta y se pueden acortar aun mas , así:

const saludar = nombre =>document.write("Hola");

***Programación Orientada a Objetos***

***Explicación básica:***

**Clases**: Es lo que crea cada tipo de objeto, si queremos crear un tipo diferente de objetos, deberíamos crear otra clase.

**Objeto**: El objeto como en la vida real es algo con atributos y que puede realizar una función.

**Atributos**: Las características que va a tener el objeto.

**Método**: Las cosas que va hacer el objeto.

Las clases se crean con un nombre y un constructor, dentro del constructor pondremos todos los parámetros que luego modificaremos para cada objeto

// Clase llamada Animal

class animal { //3 parametros

    constructor(especie,edad,color){

       this.raza = especie;            //El parámetro 1 define la especie

       this.años = edad;               //El parámetro 2 define la edad

       this.apariencia = color;        //El parámetro 3 define el color

    }

}

El ***this*** hará referencia al objeto cuando lo creemos, ósea que si creamos un objeto llamado perro, el parámetro **this.edad** pasara a ser **perro.edad** .

Métodos

// Clase llamada Animal

class animal { //3 parámetros

    constructor(especie,edad,color){

        this.raza = especie;           //El parámetro 1 define la especie

        this.años = edad;              //El parámetro 2 define la edad

        this.apariencia = color;       //El parámetro 3 define el color

    }

// Método

    verInfo(){

        document.write(this.apariencia + "<br>")

    }

}

El método es lo mismo que las funciones, pero esta adentro de la clase.

    Para llamar al método de la clase se escribe primero el objeto y

    luego el método.

perro.verInfo();

**super** se utiliza para llamar al constructor de la clase padre, en este caso, la clase **animal**. La palabra clave **super** se utiliza dentro del constructor de una clase hija para invocar el constructor de la clase padre y heredar sus propiedades.

**Método Static**

Un método estático es un método que pertenece a la clase en sí misma, en lugar de a una instancia específica de la clase. Esto significa que puedes llamar a un método estático directamente en la clase, sin necesidad de crear una instancia de la clase.

class Ejemplo {

// Método estático

static metodoEstatico() {

console.log("¡Este es un método estático!");

}

// Método de instancia

metodoDeInstancia() {

console.log("¡Este es un método de instancia!");

}

}

// Llamando al método estático sin crear una instancia

Ejemplo.metodoEstatico();

// Creando una instancia de la clase

const instancia = new Ejemplo();

// Llamando al método de instancia a través de la instancia

instancia.metodoDeInstancia();

Al método estático lo podes llamar sin necesidad de crear una instancia, es es útil cuando quieres crear un método directamente para la clase en vez de crear una instancia.

***Métodos de cadena***

Concat() Concatena una cadena con otra

startsWith() Si comienza con los caracteres de otra cadena, da true, si no false.

endsWith() Si termina con los caracteres de otra cadena, da true, si no false.

includes() Si la cadena incluye el carácter que buscamos, dará true, si no false.

indexOf() Si la cadena incluye el carácter que buscamos, nos dará el índice o posición de su primera letra, si no la encuentra dará -1.

lastIndexOf() Si la cadena incluye el carácter que buscamos, y este se repite varias veces, nos dará el índice o posición de la primera letra, de la ultima repetición, si no la encuentra dará -1.

padEnd () Rellena el inicio de la cadena hasta llegar a la cantidad de caracteres que indiquemos padStart(cantidad, “Relleno”).

padEnd () Rellena el final de la cadena hasta llegar a la cantidad de caracteres que indiquemos padStart(cantidad, “Relleno”).

repeat() Repite la cadena la cantidad de veces que especifiquemos.

split() Divide la cadena como un array, la divide cada vez que encuentre un carácter. Ej. (“Cadena de cuatro palabras”) => [0, 1, 2, 3]

substring() Recorta la cadena desde donde indiquemos hasta donde indiquemos Ej. (“Soy una cadena”) = substring(0,7) => Soy una

toUpperCase() Convierte toda la cadena a MAYUSCULAS.

toLowerCase() Convierte toda la cadena a minúsculas.

toString() Convierte números u otros caracteres a una cadena.

trim() Elimina los espacios de una cadena. Ej. (“ Santi ”)

(cadena.length) = 15

Cadena.trim();

(cadena.length) = 5 = (“Santi”)

trimEnd() Elimina los espacios del final de una cadena. Ej. (“ Santi ”)

(cadena.length) = 15

Cadena.trim();

(cadena.length) = 10 = (“ Santi”)

trimStart() Elimina los espacios del inicio de una cadena. Ej. (“ Santi ”)

(cadena.length) = 15

Cadena.trim();

(cadena.length) = 10 = (“Santi ”)

Métodos de Arrays

Transformadores

Pop() Elimina el ultimo elemento de un array y lo devuelve.

Nombres = [ “Pedro”, “Josefito”,”Recórcholis” ]

Recorte = nombres.pop ()

Recorte = Recórcholis

Nombres = Pedro, Josefito

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Shift() Elimina el primer elemento de un array y lo devuelve.

Nombres = [ “Pedro”, “Josefito”,”Recórcholis” ]

Recorte = nombres.shift ()

Recorte = Pedro

Nombres = Josefito, Recórcholis

Push() Agrega la cantidad de elementos que elijas a un array.

Nombres = [ “Pedro”, “Josefito”,”Recórcholis” ]

Push = nombres.push (“Roberta”, “Luchita”)

Push = “Pedro”, “Josefito”,”Recórcholis”, “Roberta”, “Luchita”

Nombres = 5

Reverse() Invierte el orden de los elementos de un array.

Números = [ 1, 2, 3, 4, 5 ]

numeros.reverse ()

Números = 5, 4, 3, 2, 1

Sort() Ordena los elementos de un array de menor a mayor o de A-Z.

Números = [ 5, 3, 7, 6, 4 ] Letras = [ “S” “A” “N” “T” “I”]

numeros.sort () letras.sort ()

Números = 3, 4, 5, 6, 7 Letras = A I N S T

Splice() Elimina la cantidad de elemento que indiquemos, y agregamos en su lugar los elementos que queramos.

Nombres.splice(Posición de inicio, Cantidad de elementos, “Elements”)

Números = [ 5, 3, 7, 6, 4 ] Letras = [ “S” “A” “N” “T” “I”]

numeros.sort () letras.sort ()

Números = 3, 4, 5, 6, 7 Letras = A I N S T

Map() Transforma cada elemento de un array y crear un nuevo array con los resultados de aplicar una función a cada elemento del array original.

Ej. Si tenemos un array con cadenas, y queremos mostrar cuantas letras tiene cada uno, o si tenemos un array con números y queremos mostrarlos pero duplicados por 2.

const números = [1, 2, 3, 4, 5];

// Duplica cada número del array original

const duplicados = numeros.map(numero => numero \* 2);

console.log(duplicados); // Resultado: [2, 4, 6, 8, 10]

Accesores

Join() Convierte el array en una cadena de texto. Dentro del () podemos agregar lo que queramos que esté entre medio de cada elemento.

Filter() Crea un nuevo array solo con los elementos que cumplan con alguna condición que impongamos. Como solo mostrar los elementos que tengan mas de 4 letras, etc.

let newArray = arrayOriginal.filter(function(elemento){

// Devuelve true si el elemento debe ser incluido en el nuevo arreglo, false en caso contrario

});

forEach () Es un método de los arreglos en JavaScript que permite ejecutar una función proporcionada una vez por cada elemento presente en el arreglo. La sintaxis básica es la siguiente:

array.forEach(function(elemento) {

// Acción a realizar en cada elemento

});

En resumen, *filter* se utiliza cuando se desea crear un nuevo arreglo basado en ciertos criterios de filtrado, mientras que *forEach* se utiliza cuando se quiere realizar una acción en cada elemento del arreglo sin crear un nuevo arreglo. Ambos son útiles, pero cumplen propósitos diferentes en el manejo de datos en JavaScript.

***Objeto Math***

Para aplicar los objetos math se usa el método *Math.*

Números = Math.objeto();

*sqrt*() Raíz cuadrada positiva.

*cbrt*() Raíz cubica positiva.

*max*() Recibe todos los números que le des, y devuelve el mayor.

*min*() Recibe todos los números que le des, y devuelve el menor.

*random*() Devuelve un numero pseudoaleatorio entre 0 y 1.

*round*() Redondea un número al entero más cercano.

*fround*() Redondea un número decimal a su representación de precisión simple (4byts) ósea, un numero con 15 decimales.

*floor*() Redondea un número hacia abajo al entero más cercano.

*floor*() Elimina la parte decimal de un número, dejando solo la parte entera. A diferencia de *floor* no redondea, solo elimina los decimales sin importar que sea negativo o positivo.

*PI*() Devuelve PI en precisión simple (15 dígitos).