2020

Arrays

Introducción a la Informática

JUAN DIEGO VÉLEZ SANCHEZ

NOVIEMBRE DE 2020

UTP | Pereira

1 CONTENIDO

2 PRESENTACIÓN

La presente monografía describe la implementación y utilización de los objetos de tipo arrays, que nos dan soporte al manejo, utilización e implementación de los arrays en la materia INTRODUCCIÓN A LA INFORMÁTICA.

En los siguientes párrafos se presenta una descripción básica del significado de lo que es un array en programación.

Un array también conocidos como vectores, listas, arreglos o matrices, es un objeto que nos permitirá almacenar diferentes datos de una manera ordena sencilla y eficaz.

Por ejemplo, si tendríamos que guardar en un programa 100 nombres, esto de una manera tradicional nos tomaría crear 100 variables diferentes, por lo cual no es una forma óptima para este tipo de tares ya que consumiríamos más recursos y se nos dificultaría trabajar. Para eso tenemos los arrays que nos permitirán almacenar estos 100 nombres en una sola parte.

Pero, ¿Cómo podemos manejar los datos que están contenidos en estas listas?

Como un array es un objeto, este tiene sus diferentes métodos para poder manejarlo como por ejemplo: modificarlos, agregar elementos, o quitar un elemento, revertirlos, etc.

En las siguientes secciones se explicaran y se mostraran el código de varios métodos que podemos utilizar fácilmente para utilizar de forma eficaz las listas en JavaScript.

3 CREANDO UN ARRAY

Hay diferentes maneras de crear un array en JavaScript, se debe recalcar que JavaScript es un lenguaje de programación débilmente tipado, lo cual quiere decir que no se debe poner qué tipo de dato se almacenara en una variable, también nos permite combinar en un mismo array tipo de datos diferentes como cadenas de texto, números enteros, flotantes, e incluso otros arrays.

Para crear un array se puede hacer de las siguientes maneras:

```
var miArray = new Array(3);

miArray[0] = "Juan";
miArray[1] = 1599;
miArray[2] = false;
```

Podemos ver que al crear un array simplemente debemos poner new array entre paréntesis la cantidad de elementos que va a contener (opcional). La cantidad de elementos no es lo mismo que la cantidad de posiciones, ya que si vemos guardamos diferentes elementos de acuerdo a su posición esta comienza en cero y la posición final será la dos, para acceder a un elementos del array lo hacemos de acuerdo a su índice o posición.

```
var miArray = ["Juan", 1599, false];
```

```
var miArray = new Array("Juan", 1599, false);
```

Estas también son formas validas de crear un array, si vemos los elementos están separados por comas, tiene tres elementos pero su posición máxima sigue siendo 2.

O un array sin elementos.

```
var miArray = [];
```

4 RECORRIENDO UN ARRAY

Para recorrer una array completo, simplemente podemos utilizar los diferentes bucles como el for, while, do while, forEach, que tenemos en el lenguaje de programación JavaScript.

```
for (var i = 0; i < myArray.length; i++){
    myArray[i] = .....

    //cualquier codigo...
}
```

Si vemos utilizamos una variable que se incrementara en cada iteración, esta tendrá un máximo dependiendo de la cantidad de elementos, menos 1, ya que la cantidad de posiciones es uno menor que la cantidad de elemento en el array.

```
var myArray = ["Juan", 1599, false];

for (var i = 0; i < myArray.length; i++){
    document.write(myArray[i] + "<br/>");
}
```

La salida será:

Juan

1599

False

El método length nos devuelve un número entero este dependerá de la cantidad de elementos del array.

También podemos utilizar el forEach como método del array, forEach ejecuta una función indicada una vez por cada elemento del array.

```
myArray.forEach(function(elemento){
    document.write(elemento + "<br/>");
});
```

Se crea una función interna y su argumento es el elemento de acuerdo a su posición, esto permite recorrer el array completamente. Puede tener más de un argumento de acuerdo a su utilidad, en el ejemplo imprimimos todos los elementos del array: Juan, 1599, False.

5 AGREGANDO Y ELIMINANDO ELEMENTOS

Para agregar y quitar elementos en un array es muy fácil, podemos utilizar diversos métodos como los siguientes, estos métodos cambian la longitud del array.

```
var myArray = ["Juan", 1599, false];

myArray.push("Hola Mundo");
```

Push() es un método que nos permite agregar un elemento al final del array, en este ejemplo agregamos "Hola Mundo", el modifica la cantidad de elementos que contiene, podemos agregar cualquier cosa con este método.

```
myArray.pop();
```

Pop() eliminara el último elemento en el array, como utilizamos **push()** para poner "Hola Mundo" como último elemento, al utilizar **pop()** este eventualmente se eliminara.

```
myArray.unshift("Hola Mundo");

myArray.shift();
```

Unshift() tiene la misma funcionalidad que **push(),** sin embargo esta nos permite agregar el elemento en la primera posición, al igual que **pop()** que permite eliminar el último elemento, shift() hace lo contrario eliminando el primer elemento del array. Para insertar o eliminar elementos en una posición definida podemos hacerlo de la siguiente manera:

```
var myArray = ["Juan", 1599, false];
```

```
var nuevosElemetos = ["Java", "HTML", true];

myArray.splice(2, 0, nuevosElemetos);
```

Esto nos permite agregar los elementos "Java", "HTML", "true", después del segundo elemento del array. El array es **[Juan, 1599, Java, HTML, true, false].** Para eliminar solo quitamos el tercer parámetro, el primer argumento es el inicio y el segundo el final.

6 CONCATENAR Y CREAR UN ARRAY APARTIR DE OTRO

```
let vegetales = ['Repollo', 'Nabo', 'Rábano',
'Zanahoria'];

// ["Repollo", "Nabo", "Rábano", "Zanahoria"]

let elementosEliminados = vegetales.splice(1, 2);

/* ["Nabo", "Rábano"] ==> Lo que se ha guardado en
"elementosEliminados" */

// ["Repollo", "Zanahoria"] ==> Lo que actualmente
tiene "vegetales"
```

Creamos un array con algunos nombres de vegetables como sus elementos, podemos ver que se crea una nueva variable, que contiene el resultado del método **splice(),** lo que hace este método en esta ocasión, es cortar los elementos desde una posición de inicio y fin, en el anterior ejemplo se toman los elementos desde la posición 1 hasta la 2, si recordamos el primer elemento (0) es "Repollo". Hay que recalcar que si utilizamos este método de esta forma el array principal también es modificado, eliminando los elementos dados sus posiciones.

```
let copiaArray = vegetales.slice();

// ["Repollo", "Zanahoria"]; ==> Copiado en
"copiaArray"
```

Para copiar un array simplemente basta son el método **slice(),** hace una copia exacta del array original.

El método **concat()** se usa para unir dos o más arrays. Este método no cambia los arrays existentes, sino que devuelve un nuevo array.

```
const array1 = ['a', 'b', 'c'];
const array2 = ['d', 'e', 'f'];
const array3 = array1.concat(array2);

// Nuevo Array ["a", "b", "c", "d", "e", "f"]
```

7 ENCONTRAR, OBTENER INDICE DE UN ELEMENTO EN EL ARRAY

Para saber si un elemento esta en un array podemos recorrerlo con un bucle y verificar cada elemento si es igual al deseado, sin embargo también podemos utilizar la siguiente opción.

```
var nombres = ["David", "Sofía", "Ramón", "Carlos",
"María"];

var posicion = nombres.indexOf("Carlos");
```

En el anterior código podemos ver que tenemos un array de 5 elementos que contienen cadenas de texto, si vemos una variable almacenara el resultado del método **indexOf(),** este nos devuelve un número entero que es el índice donde se encuentra tal elemento. Si el elemento no se encuentra devuelve un -1, si el elemento se repite devolverá la primera posición en donde se encuentra. El elementos a buscar es “Carlos” como este se encuentra en el array nos devuelve un 3 ya que este es el cuarto elemento.

```
var nombres = ["David", "Sofía", "Ramón", "Carlos",
"María", "Carlos"];

var posicion = nombres.lastIndexOf("Carlos");
```

Con este método podemos saber la última posición donde se encuentra el elemento a buscar, podemos ver que ”Carlos” se encuentra en la posición 3 y 5, como queremos la última posición en este caso nos retornara el número 5 que es la última posición donde se encuentra tal elemento.

```
const array1 = [5, 12, 8, 130, 44];

const found = array1.find(element => element > 10);

// found == 12
```

El método **find()** devuelve el valor del primer elemento del array que cumple la función de prueba proporcionada.

El método **includes()** determina si una matriz incluye un determinado elemento, devuelve true o false según corresponda.

8 SEPARANDO Y UNIENDO ELEMENTOS

```
var frase = "Los diferente metodos de los arrays en
JavaScript";

var palabras = frase.split(" ");
```

Palabra: 0: Los
Palabra: 1: diferente
Palabra: 2: metodos
Palabra: 3: de
Palabra: 4: los
Palabra: 5: arrays
Palabra: 6: en
Palabra: 7: JavaScript

El método **split()** nos permite separar los elementos de acuerdo a un carácter, la variable palabras se convertirá en un array ya que cada elemento esta separado por un espacio, o sea el valor del array será las diferente palabras que le proceden un carácter en blanco o vacío, la palabra "Los" será el primer elemento, "diferente" el segundo y así sucesivamente hasta la palabra "JavaScript" que será la última. Para separar algo cuando se encuentre una coma, como parámetro del método **Split()** le introducimos tal carácter.

```
var words = ["Los", "diferentes", "metodos", "de",
"los", "arrays", "en", "JavaScript"];

var text = words.join();
```

Gracias al método **join()** podemos unir un array en una sola variable el resultado del anterior código es:

Los,diferentes,metodos,de,los,arrays,en,JavaScript

Los elementos son separados por comas.

9 INVERTIR Y ORDENAR UN ARRAY

Para invertir un array simplemente utilizamos el método reverse.

```
var numeros = [1,2,3,4,5,6,7,8,9,10];

numeros.reverse();

// 10,9,8,7,6,5,4,3,2,1
```

¿Qué tal si después de invertirlos los ordenamos?.

```
numeros.sort();

// 1,10,2,3,4,5,6,7,8,9
```

El método **sort()** ordena los elementos de un array, además devuelve el arreglo ordenado. Podemos ver que se ordena los elementos, sin embargo si miramos cuidadosamente, los elementos se ordenan dependiendo de su primer valor o carácter, primero se ordenaron aquellos que comienza con el número uno (1).

```
numeros = [10,1,20,2,43,5,11,50];

numeros.sort();

// 1,10,11,2,20,43,5,50
```

Vemos que no se ordenan de modo ascendente de nuevo [1, 2, 5, 10, 11, 20, 43, 50]. SI no nuevamente de acuerdo a su primer carácter. Para evitar esto podemos pasarle como parámetro de método una función que nos permitirá ordenarlo de forma correcta como la siguiente:

```
function comparar(a, b) {
    return a - b;
}

numeros.sort(comparar);
```

Gracias a la función que creamos se ordenaran de modo ascendente su resultado será: 1,2,5,10,11,20,43,50

```
var letras = ["Z", "D", "C", "B", "A", "U"];

letras.sort();

// A,B,C,D,U,Z
```

```
var cosas = ["HOLA", "HUNGRIA", "ALAS", "AVES"];

cosas.sort();

// ALAS,AVES,HOLA,HUNGRIA
```

Las palabras son ordenas alfabéticamente.

```
var numerosLetras = ["A", 1, 2, 3, "B", "C", 4, "D", 5
, "E"];

numerosLetras.sort();

// 1,2,3,4,5,A,B,C,D,E
```

Si ordenamos un array que contiene tanto números como letras, JavaScript ordena primero los elementos numéricos, y luego los elementos de texto.

10 OTROS METODOS

Array.isArray(*myArray*) Retornara true o false si aquella variable es un contendor de elementos.

```
var es_un_Array = [1,2,3];

Array.isArray(es_un_Array);

// True
```

```
var es_un_Array = "Hola Mundo";

Array.isArray(es_un_Array);

// False
```

El método toString() devuelve una cadena de caracteres representando el array especificado y sus elementos.

```
const array1 = [1, 2, 'a', '1a'];

array1.toString();

// "1,2,a,1a"
```

Array.length Devuelve la cantidad de elementos de un array pero sus posiciones es un número menor que el orden de los elementos, o sea sus posiciones comienzan desde cero (0).

```
const array1 = [1, 2, 'a', '1a'];
var cantidad = array1.length;

// Cantidad = 4
```

```
array1.includes(1);
```

Este método retorna true o false si el elemento entre los paréntesis se encuentra en el array.

11 FUNCIONES, ITERADORES EN LOS ARRAYS

En esta sección mostraremos, aquellos métodos que tiene como argumentos el llamado a una función, los métodos que vamos a ver recorren todo el array y cada elemento es evaluado de acuerdo a una función.

Anteriormente ya hemos visto una forma de utilizar estos métodos, el cual era el **forEach(),** que permite recorrer todo el array completo y además llamar una función, o crear una función interna.

El método **some()** comprueba si al menos un elemento del array cumple con la condición implementada por la función proporcionada.

```
function esPar(numero) {
    return numero % 2 == 0;
}

var numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
var algunoPar = numeros.some(esPar);

// True
```

Lo que hace internamente es coger el primer elemento (1) y llama a la función esPar(), esta nos devuelve true o false, después coge el segundo elemento hasta el último que es el número 10 en este caso.

```
function esLetra(e) {
    return isNaN(e);
}

var arg = [1,2,3,4,5];

arg.some(esLetra);

// False
```

En este caso se verifica si algún elemento es de tipo texto, como no hay ningún elemento de tipo texto retornara false.

Una forma de convertir un array en un único valor podemos utilizar el método **reduce()**. El método **reduce()** ejecuta una función reductora sobre cada elemento de un array, devolviendo como resultado un único valor.

La función genera un ciclo oculto que recorre todo el array, además recibe cuatro argumentos.

1. Acumulador (acc)
2. Valor Actual (cur)
3. Índice Actual (idx)
4. Array (src)

El valor actual de la función se almacena al acumulador en cada iteración, y en la última instancia o iteración se convierte en el valor final y único. Anteriormente hemos visto los elementos de este método, el único elemento opcional y que podemos descartar es el tercero.

Para todos estos métodos que hemos visto podemos utilizar una función ya definida o crear una interna que solo servirá para una ocasión. Por ejemplo concatenar los elementos de un array, esta se almacena en una variable.

```
var unirArray = ["Funciones", "iteradores", "en",
"los", "arrays"];

var union = unirArray.reduce(function(acc, cur){
    return acc + " " + cur;
});

// union == Funciones iteradores en los arrays
```

En este caso hemos utilizado una función interna, esta no solo se ejecuta en esta parte del código y no se puede llamar en otros métodos ya que no está definida, como podemos ver se ha utilizado tres elementos esenciales de este método, que es el array al que le queremos reducir (unirArray), el elemento actual (acc) y el acumulador (cur), en esta ocasión no es necesario el índice actual. También podemos utilizar el método llamando a una función, pero esta debe tener por lo menos como parámetros de función el valor actual y el acumulador.

Este método **reduce(),** comienza de izquierda a derecha lo cual si queremos invertirlo, o sea de derecha a izquierda, podemos utilizar el método **reduceRight().**

```
var unirArray = ["Funciones", "iteradores", "en",
"los", "arrays"];

var union = unirArray.reduceRight(function(acc, cur){
    return acc + " " + cur;
});

// union == arrays los en iteradores Funciones
```

Los mismos elementos de texto sin embargo están invertidos.

El siguiente método nos permitirá modificar cada elemento del array, este devolverá un nuevo array con los elementos modificado de acuerdo a una función.

El método **map()** crea un nuevo array con los resultados de la llamada a la función indicada aplicados a cada uno de sus elementos.

```
var arrayMap = [1 , 2 , 3, 4, 5];

arrayMap = arrayMap.map(function(elemento){
    return elemento.toString() + " Elemento";
});


// arrayMap ==> 1 Elemento,2 Elemento,3 Elemento,4
Elemento,5 Elemento
```

La instrucción **toString()** convierte números a cadenas de texto, lo cual los números almacenados en el array se convierten en texto pero además son concatenados con la palabra " Elemento", al final tenemos un nuevo array, ya no de números sino de cadenas de texto.

Si queremos crear un nuevo array a partir de otro, pero este nuevo array conteniendo los elementos de acuerdo a una condición podemos utilizar el método **filter().**

filter() crea un nuevo array con todos los elementos que cumplan la condición implementada por la función dada.

```
var numbers = [1,2,3,4,5,6,7,8,9,10];

numbers = numbers.filter((elemento) => {
    return elemento % 2 === 0;
});

// numbers ==> 2,4,6,8,10
```

Esta también es una forma de crear una función que solo se utilizara una vez, es llamada función flecha, los parámetros de esta función son introducidos entre paréntesis y seguido del igual (=) y mayor que (>), (=>) entre las llaves el contenido de la función. En el anterior ejemplo cada elemento del array será evaluado si es par, de ser así se guardara en el nuevo array.

12 ARRAYS MULTIDIMENSIONES

Un array multidimensional es aquel que en su interior contiene elementos de tipo array.

Pensemos en una lista cada ítem representa cada elemento del array, pero que tal si en cada ítem tenemos una lista esto sería como tener subitems.

1. Item A
    a. Subitem 1
    b. Subitem 2
2. Item B
3. Item C
    a. Subitem 1
    b. Subitem 2
4. Item D

O también como columnas y filas, donde las filas son cada array, como los ítems de la lista y las columnas son los elementos de cada array o los subitems de la lista.

array 1	Elemento	Elemento	Elemento
array 2	Elemento	Elemento	Elemento
array 3	Elemento	Elemento	Elemento
array 4	Elemento	Elemento	Elemento
array ...	Elemento	Elemento	Elemento

Las formas de crear un array de este tipo son las siguientes:

```
var arrayMulti = new Array(2);

arrayMulti[0] = new Array(2);
arrayMulti[1] = new Array(2);

arrayMulti[0][0] = "HTML";
arrayMulti[0][1] = "CSS";

arrayMulti[1][0] = "JAVASCRIPT";
arrayMulti[1][1] = "C++";
```

Se crea un array bidimensional, contiene dos elementos de tipo array y además estos contienen dos elementos. Se puede apreciar que para acceder a cada posición del array se utilizan las posiciones entre corchetes, el primer corchete sirve para acceder a los elementos del array original, es equivalente a los ítems de una lista, el segundo corchete es equivalente a los subitems de una lista.

```
var array = [
    ["HTML", "CSS"],
    ["JAVASCRIPT", "C++"],
    ["Visula Studio", "Pycharm", "Eclipse"]
];
```

Esta también es una forma de crear un array bidimensional, el array original tiene 3 elementos, y sus subelementos son los que están entre los corchetes, para acceder a estos se utiliza la misma sintaxis que vimos en la primera forma.

Solo hemos visto arrays bidimensionales pero también podemos tener tridimensionales incluso más de una dimensión, sin embargo es difícil trabajar con ellos.

Ejemplo array tridimensional:

```
var tridimensional = [

    // inicio primer array
    [
        ["SubArray1.1", "elemento", "elemento"],
        ["SubArray1.2", "elemento", "elemento"],
        ["SubArray1.3", "elemento", "elemento"],
    ],
    // Fin primer array

    // inicio segundo array
    [
        ["SubArray2.1", "elemento", "elemento"],
        ["SubArray2.2", "elemento", "elemento"],
        ["SubArray2.3", "elemento", "elemento"],
    ]
    // Fin segundo array

];

alert(tridimensional[0][0][0]);

alert(tridimensional[1][0][0]);
```

Alert() perimirá mostrar un mensaje, en nuestro caso los elementos en tal posición del array, como es un array tridimensional utilizamos tres corchetes para acceder a sus elementos.

El array base tiene dos elementos, el primer subArray, por así llamarlo, tiene también tres elementos estos también de tipo array, los cuales contienen elementos de tipo texto, igualmente para el subArray 2.

Lastimosamente JavaScript no tiene muchas funcionalidades como métodos para trabajar con estos tipos de arrays, lo cual al final son difíciles de utilizar.

13 RECORRER ARRAYS MULTIDIMENSIONES

Para recorrer este tipo de arrays se deben utilizar bucles anidados, o sea un bucle dentro de otro, además hay que tener en cuenta con que array se trabaja ya que dependiendo de sus dimensiones, es necesario la misma cantidad de bucles anidados.

Para un array normal de una dimensión es muy simple.

```
var myArray = ["Juan", 1599, false];

for (var i = 0; i < myArray.length; i++){
    document.write(myArray[i] + "<br/>");
}
```

Imprime en el documento los elementos de array.

Para un array bidimensional se utilizan dos bucles, el primero recorre los elementos del array original, y el segundo los elementos de los subArrays, como si fueran ítems y subitems de una lista.

```
var array = [
    ["HTML", "CSS"],
    ["JAVASCRIPT", "C++"],
    ["Visula Studio", "Pycharm", "Eclipse"]
];

for(var i = 0; i < array.length; i++){
    for(var j = 0; j < array[i].length; j++){
        document.write(array[i][j] + "<br/>");
    }
}
```

Para arrays tridimensionales utilizamos tres bucles, nos imprime los elementos del tercer array.

```
var tridimensional = [

    // inicio primer array
    [
        ["SubArray1.1", "elemento", "elemento"],
        ["SubArray1.2", "elemento", "elemento"],
        ["SubArray1.3", "elemento", "elemento"],
    ],
    // Fin primer array

    // inicio segundo array
    [
        ["SubArray2.1", "elemento", "elemento"],
        ["SubArray2.2", "elemento", "elemento"],
        ["SubArray2.3", "elemento", "elemento"],
    ]
    // Fin segundo array

];

for(var i = 0; i < tridimensional.length; i++){

    for(var j = 0; j < tridimensional[i].length; j++){

        for(var k = 0; k < tridimensional[i][j].length; k++){

            document.write(tridimensional[i][j][k] + "<br/>");

        }
    }
}
```

Y si sucesivamente si tiene más arrays anidados.

14 UNIFICAR ARRAYS

Si tenemos un array bidimensional lo podemos unir todos los elementos en un solo array.

El método **flat()** crea una nueva matriz con todos los elementos de sub-array concatenados recursivamente hasta la profundidad especificada.

Recibe como parámetro un número que indica que profundidad queremos unir como en este caso es un array bidimensional no introducimos nada ya que el valor predeterminado es 1.

```
var array = [
    ["HTML", "CSS"],
    ["JAVASCRIPT", "C++"],
    ["Visula Studio", "Pycharm", "Eclipse"]
];

array = array.flat();

// array ==> HTML,CSS,JAVASCRIPT,C++,Visula Studio,Pycharm,Eclipse
```

Para un array tridimensional se hace lo siguiente:

```
var tridimensional = [

    // inicio primer array
    [
        ["SubArray1.1", "elemento", "elemento"],
        ["SubArray1.2", "elemento", "elemento"],
        ["SubArray1.3", "elemento", "elemento"],
    ],
    // Fin primer array

    // inicio segundo array
    [
        ["SubArray2.1", "elemento", "elemento"],
        ["SubArray2.2", "elemento", "elemento"],
        ["SubArray2.3", "elemento", "elemento"],
    ]
    // Fin segundo array

];
```

```
var arrayUnificado = tridimensional.flat(2);
```

Obtenemos un nuevo array con 18 elementos

```
arrayUnificado.forEach((elemento) => {
    document.write(elemento + "<br/>");
});
```

Si los escribimos en el documento obtenemos

SubArray1.1
elemento
elemento
SubArray1.2
elemento
elemento
SubArray1.3
elemento
elemento
SubArray2.1
elemento
elemento
SubArray2.2
elemento
elemento
SubArray2.3
elemento
elemento

Y así es como unificamos los arrays multidimensionales, toca recalcar que la profundidad es uno menos de los array anidados, si el array es tridimensional la profundidad es 2, si es cuatridimensional la profundidad es 3, y así sucesivamente.

15 OBJETOS, ATRIBUTOS Y METODOS

Hemos dicho que un array es un objeto, pero ¿Qué es en realidad un objeto en programación?

La programación orientada a objetos es un paradigma de programación, ejemplo la manera más sencilla que utilizamos es el paradigma secuencial, las instrucciones van una tras otras, de arriba abajo.

Un objeto en programación no es nada más que un contenedor de variables, y métodos. Ejemplo:

Un automóvil lo podemos llamar un objeto, este comparte características y funcionalidades, todos los automóviles tiene ruedas, una velocidad, puertas, puede tener aire acondicionado, etc. Pero también tiene sus funcionalidades como lo es encender, arrancar, girar etc. Las características las podemos llamar variables de objeto que son los datos de estos, comúnmente llamados atributos y sus funcionalidades son los métodos.

```
var auto = {
    marca: "Audi",
    año: "2002",
    velocidad: 290,
    electrico: false
};

var auto2 = {
    marca: "Audi",
    año: "2002",
    velocidad: 290,
    electrico: false
};

alert(auto.marca);
alert(auto.año);
alert(auto.velocidad);
alert(auto.electrico);
```

Para crear un objeto basta con poner sus variables entre paréntesis, dato y valor separados por comas, para acceder a estos datos copiamos el nombre del objeto y con un punto seguido del nombre del dato al que queremos acceder.

Sin embargo no es una forma muy útil ya que si queremos replicar este objeto con diferente valores, debemos hacer lo mismo lo cual no es recomendado. Los objetos en JavaScript los

podemos crear con funciones ya que así podemos crear objetos inmediatamente en pocas líneas de código.

```
function Auto(marca, año, velocidad, electrico){
    this.marca = marca;
    this.año = año;
    this.velocidad = velocidad;
    this.electrico = electrico;
}

var myAuto = new Auto("Audi", "2002", 290, false);

var segundoCoche = new Auto("Tesla", "2020", 300, true);
```

Vemos que para crear un objeto con funciones debemos poner la palabra **new** seguido del nombre de la función, podemos tener tantos parámetros como queramos, estos serán los atributos de nuestro objeto, por eso utilizamos la instrucción (**this**) que nos permite almacenar como variables, es lo mimo a tener dato y valor, para acceder a estos simplemente hacemos lo mismo nombre de objeto y seguido de punto el atributo que queremos.

```
myAuto.marca

// Audi

myAuto.velocidad

// 290
```

Ahora creamos un método que nos permitirá cambiar la velocidad del auto.

```
function Auto(marca, año, velocidad, electrico){
    this.marca = marca;
    this.año = año;
    this.velocidad = velocidad;
    this.electrico = electrico;

    this.changeVelocidad = function(velocidad){
        this.velocidad = velocidad;
    }
}
```

El método es una función interna que reescribe el atributo velocidad, si lo utilizamos obtenemos lo siguiente.

```
var myAuto = new Auto("Audi", "2002", 290, false);

myAuto.marca;
// Audi

myAuto.velocidad;
// 290

myAuto.changeVelocidad(350);

myAuto.velocidad;
// 350
```

Como el código se ejecuta de arriba hacia abajo, la velocidad inicial es 290 pero luego es cambiada con el método que creamos, su nuevo valor será 350. Podemos crear todos los métodos que queramos.

```
function Auto(marca, año, velocidad, electrico){
    this.marca = marca;
    this.año = año;
    this.velocidad = velocidad;
    this.electrico = electrico;

    this.changeMarca = function(marca){
        this.marca = marca;
    }

    this.changeAño = function(año){
        this.año = año;
    }

    this.changeVelocidad = function(velocidad){
        this.velocidad = velocidad;
    }

    this.changeElectrico = function(electrico){
        this.electrico = electrico;
    }
}
```

Ya podemos instanciar todos los objetos autos que queremos y modificar sus atributos por medio de los métodos. Podemos agregar más atributos y métodos.

La programación orientada a objetos es el paradigma de programación más utilizado en el mundo y se base en los siguientes cuatro pilares:

1. Abstracción.
2. Encapsulamiento.
3. Polimorfismo.
4. Herencia.

Una simple definición de lo anterior es:

Abstracción: Atributos y métodos.

Encapsulamiento: Proteger la información para no manipularlos, datos privados, públicos, métodos públicos privados, etc.

Polimorfismo: Darles la misma orden a cada objeto y que cada uno responda de su propia manera.

Herencia: La herencia es como en la vida real, ejemplo los hijos heredan genes de los padres, pero estos no son copias exacta, tienen algunos rasgos propios, en los objetos es igual, los objetos heredan atributos y métodos, además tienen sus propios atributos y métodos.

16 BIBLIOGRAFÍA

https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/Array