JAVASCRIPT

CLASE 6
Material complementario

ARRAYS

CODER HOUSE

JavaScript: arrays

Arrays: definición

Los ***arrays*** (también llamados ***colecciones***) son un tipo de objeto especial que nos permite agrupar elementos. Se utilizan cuando necesitamos contar con un conjunto de valores asociados a un listado. En estas estructuras, podemos guardar cualquier tipo de dato u objeto, es decir que es posible crear arrays de números, cadenas, booleanos, objeto e incluso un array compuesto por otros arrays (matrices).

Si bien el array es un objeto, para crear una estructura de este tipo no es necesario emplear el constructor (por ejemplo: new Array()), sino que nos basta con usar los corchetes y especificar los elementos que componen la lista separados por coma (,), como podemos observar en el ejemplo:

```
// Declaraciòn de array vacío
const arrayA = [];
// Declaración de array con nùmeros
const arrayB = [1,2];
// Declaracion de array con strings
const arrayC = ["C1","C2","C3"];
// Declaración de array con booleanos
const arrayD = [true,false,true,false];
// Declaración de array mixto
const arrayE = [1,false,"C4"];
```

Dado que el array es un objeto, es preferible declarar la variable a la que se asigna esta estructura con la palabra reservada ***const***, para evitar una posible sobre-escritura de la la referencia.

Por otro lado, es importante determinar que los arrays pueden contener elementos heterogéneos, siendo necesario un control por parte del/la programador/a sobre el tipo de elemento a incluir en la colección, con la intención de evitar problemas de procesamiento al momento de operar con elementos de distinto tipo.

Si el array es homogéneo, es decir, todos sus elementos tienen el mismo tipo de dato, no se presenta inconveniente alguno al momento de hacer operaciones entre los valores de la lista. Podemos observar esto en el siguiente array numérico:

```
const  numeros = [1,2,3,4,5];
let resultado1  = numeros[0] + numeros[2]; // 1 + 3 = 4;
let resultado2  = numeros[1] + numeros[4]; // 2 + 5 = 7;
let resultado3  = numeros[1] + numeros[1]; // 2 + 2 = 4
```

Para acceder a un elemento de array, tenemos que emplear su posición o índice. Dado que los elementos de estas estructuras tienen un orden, para el ejemplo podríamos identificarlo a simple vista de la siguiente manera;

- 1 es el primer elemento del array.
- 2 es el segundo elemento del array.
- 3 es el tercer elemento del array.
- 4 es el cuarto elemento del array.
- 5 el el quinto elemento del array.

Lo anterior sería una interpretación incompleta para determinar las posiciones reales de los elementos en la lista, porque en los arrays las posiciones inician desde el número 0. Entonces, para identificar el ordenamiento correcto decimos:

- 1 está en la posición 0 del array.
- 2 está en la posición 1 del array.
- 3 está en la posición 2 del array.
- 4 está en la posición 3 del array.

- 5 está en la posición 4 del array.

Ahora que tenemos correctamente identificadas las posiciones de los elementos, para acceder a cada uno empleamos el identificador del array seguido de corchetes, y las posiciones entre estos; por ejemplo: numeros[0] ,numeros[1] ,numeros[2], numeros[3] y numeros[4].

También es posible emplear la estructura ***for*** para acceder a un elemento del array en cada ciclo. A esta forma de acceso a los elementos se la conoce con el nombre de ***"recorrido del array"***:

```
const numeros = [1, 2, 3, 4, 5];
for (let index = 0; index < 5; index++) {
    alert(numeros[index]);
}
```

Podemos acceder a todos los elementos en este caso porque hacemos coincidir el valor de la variable de conteo (index) con una posiciòn válida del array. Como el hasta (index < 5) implica iterar hasta que index tenga un valor igual a 4, y de 0 a 4 existen posiciones en la lista, se puede obtener elemento a elemento sin inconvenientes. En el caso de que quiera acceder con un índice de valor 5 al array, y no existe un elemento para dicha posición, se obtiene undefined

Métodos comunes

Como todo objeto en JavaScript, array tiene propiedades y métodos que podemos utilizar para solucionar ciertas situaciones, hacer transformaciones de los valores o realizar búsquedas y filtros.

En esta secciòn presentaremos los mètodos màs utilizados. Para conocer otras herramientas pueden verificar la especificación del objeto array en la documentación.

toString

El método toString convierte un array en un string, compuesto por cada uno de los elementos del array separados por comas. Es muy útil cuando queremos incluir el detalle de los elementos del array en una salida (console.log o alert). No obstante, hay que tener en cuenta que esta transformación no se realiza adecuadamente cuando hacemos ***toString*** a un array de objetos personalizados. En el siguiente ejemplo vemos cómo podemos emplear el métodos ***toString*** para obtener una cadena desde una valor mixto:

```
const miArray = ["marca", 3 ,"palabra"];
console.log( miArray.toString() );//imprime "marca,3,palabra"
```

Push

Este método se utiliza para sumar un elemento a un array ya existente, pasando como parámetro el valor (o variable) a agregar. Dicho elemento se agregaría al final del listado, pudiendo enviar cualquier tipo de dato como parámetro al método.

Al igual que en un string, la propiedad length nos sirve para obtener el largo de un array, es decir, cuántos elementos tiene. Cada vez que realizamos push de un elemento, el length del array aumenta en uno:

```
const miArray = ["marca", 3, "palabra"];
miArray.push('otro elemento');
console.log(miArray.length);//El array ahora tiene 4 posiciones
```

Join

Mediante este método podemos juntar todos los elementos de un array en una cadena string, indicando como parámetro el separador para esos elementos. Al igual que ***toString(),*** la nueva cadena creada comprende todos los elementos del array, con la diferencia de que en vez de estar separados por coma, se utiliza la subcadena por parámetro (o delimitador) para separar los elementos en la cadena resultante:

```
const otroArray = ["hola", 22, "mundo"];
console.log(otroArray.join("*")); //Imprime "hola*22*mundo"
```

Hay que tener en cuenta que esta transformación, al igual que en toString, no se realiza adecuadamente cuando hacemos join a un array de objetos personalizados.

Concat

A veces tenemos la necesidad de unir dos arrays, es decir, crear un nuevo array desde los elementos de dos arrays previamente declarados. Esto lo podemos hacer mediante el método concat, que nos retorna un único array compuesto por los elementos de ambos.

```
const miArray    = ["ford", 45];
const otroArray  = ["hola", 22, "mundo"];
const nuevoArray = miArray.concat(otroArray);
// nuevoArray ahora es igual a[ford,45,hola,22,mundo]
```

Es importante identificar que llamamos al método concat desde un array pasando por parámetro la segunda lista. Esta operación no es conmutativa, lo cual implica que los elementos del array desde el cual se llama al método estarán primero que los del array enviado por parámetro.

Slice

Este método retorna un nuevo array, que representa una copia de una parte de otro. Como parámetros, se envían la posición de inicio y la final, valores que se utilizarán para determinar qué elementos incluimos en la copia. Veamos un ejemplo:

```
const nombres = ['Rita', 'Pedro', 'Miguel', 'Ana', 'Vanesa'];
const masculinos = nombres.slice(1, 3);//Nuevo array desde la
posición 1 a 3.
// masculinos contiene ['Pedro','Miguel']
```

En este caso, el nuevo array se genera desde la posición 1 a la 3, pero lo que hay que tener en cuenta es que la segunda posición enviada no se incluye en el array retornado. Es decir que el valor en la posición 3 no forma parte de la copia,

Hay que observar que en ningún momento se está modificando el array nombres, sino que se crea un nuevo array que asignamos a la variable masculinos. Pero el array original sigue conteniendo los mismo valores. Para eliminar un elemento de un array podemos usar el mètodo splice, o el operador delete

Usando métodos de array

La mayoría de los métodos de array abordados en la sección anterior se enfocan en generar nuevos resultados basados en la informaciòn del array: strings que detallan los elementos de la lista con un separador, la unión de dos array, o un nuevo array más pequeño que el original.

Es necesario realizar estas operaciones con la intención de generar entradas y salidas adecuadas. Veamos un ejemplo de cómo los métodos de array nos permiten enriquecer la información que brindamos al usuario:

```
//Declaraciòn de array vacío y variable para determinar cantidad
const listaNombres = [];
let   cantidad     = 5;
//Empleo de do...while para cargar nombres en el array por prompt()
do{
   let entrada = prompt("Ingresar nombre");
   listaNombres.push(entrada.toUpperCase());
   console.log(listaNombres.length);
}while(listaNombres.length != cantidad)
//Concatenamos un nuevo array de dos elementos
const nuevaLista = listaNombres.concat(["ANA","EMA"]);
//Salida con salto de línea usando join
```

```
alert(nuevaLista.join("\n"));
```

En el ejemplos se emplean las siguientes herramientas del objeto array para añadir funcionalidad:

- Usamos la propiedad length del array para determinar cuántos elementos tenemos cargados.
- El método push nos permite añadir al array la entrada ingresada por el usuario, previa transformación del string a mayúscula con su método toUpperCase.
- Con concat creamos una nueva colección con dos elementos adicionales. El parámetro puede ser un array creado de forma explícita: [“ANA”, “EMA”];
- El método join nos permite agregar el mensaje al alert, empleando "\n" como delimitador, para generar un salto de línea entre cada elemento del array, otorgando así al usuario una salida más apropiada para un listado de nombre.

Arrays de objetos

Mencionamos en la sección anterior que existen arrays de objetos personalizados, que son un listado de objetos creados por el usuario, ya sea de forma explícita usando objetos literales, o instanciándolos con un constructor. Veamos un caso de declaración con objetos literales:

```
const objeto1 = { id: 1, producto: "Arroz" };
const array   = [objeto1, { id: 2, producto: "Fideo" }];
array.push({ id: 3, producto: "Pan" });
```

Los arrays de objeto a veces implican un tratamiento especial. Como vimos, toString y join son métodos que no funcionan correctamente para viauslizar su información, y esto se debe a que cuando buscamos transformar la información de un objeto a string, por defecto se obtiene la cadena [object Object]. Si bien más adelante evaluaremos técnicas de transformación para obtener correctamente la información de un objeto en formato

string, la primera herramienta que se puede utilizar para acceder a la información de cada objeto en un array es el bucle ***for...of***, que permite realizar el recorrido de los objetos.

Recorriendo un array de objetos

La sentencia ***for...of*** permite recorrer un objeto iterable (array), ejecutando un bloque de código por cada elemento del objeto. Para el caso de un array de objeto, la declaración de dicha estructura podría tener la siguiente forma:

```
const productos = [{ id: 1, producto: "Arroz" },
                   { id: 2,  producto: "Fideo" },
                   { id: 3,  producto: "Pan" }];

for (const producto of productos) {
    console.log(producto.id);
    console.log(producto.producto);
}
```

Por cada ciclo del ***for...of*** obtenemos la referencia a un elemento del array en orden, es decir que accedemos de uno en uno a todos los elementos del array. El bucle termina una vez iterado el último de la lista. La referencia al elemento actual la tenemos en la variable creada antes de la palabra reservada ***of***.

Como es un array de objetos, podemos acceder a su estructura con las dos formas de acceso: identifcador.propiedad (Ejemplo producto.id) o identificador[“propiedad”] (por ejemplo producto[“id”]).

Si el objeto tiene comportamiento, podemos emplear ***for...of*** para llamar a uno o más métodos pertenecientes a los objetos que componen el array, como podemos ver a continuación:

```
class Producto {
    constructor(nombre, precio) {
        this.nombre   = nombre.toUpperCase();
        this.precio   = parseFloat(precio);
        this.vendido = false;
    }
    sumaIva() {
        this.precio = this.precio * 1.21;
    }
}
//Declaramos un array de productos para almacenar objetos
const productos = [];
productos.push(new Producto("arroz", "125"));
productos.push(new Producto("fideo", "70"));
productos.push(new Producto("pan", "50"));
//Iteramos el array con for...of para modificarlos a todos
for (const producto of productos)
    producto.sumaIva();
```

También podemos anidar un for...in en el ***for...of*** para recorrer todas las propiedades del objeto actual. Además, es posible emplear las ya conocidas sentencias break o continue con ***for...of***.

Métodos de búsqueda y transformación

Si analizamos una aplicación web actual, por ejemplo una tienda virtual, notaremos que buscar y filtrar un grupo de elementos es algo común. Para implementar estas funcionalidades, tenemos en JavaScript algunos métodos de array que nos ahorran la codificación desde cero de este comportamiento. En los próximos párrafos veremos algunos de los métodos más importantes para buscar y transformar un array.

Find

El método ***find()*** devuelve el valor del primer elemento del array que satisface la función de comprobación enviada por parámetro. Si ningún valor satisface la función de comprobación, se devuelve undefined:

```
const numeros    = [1, 2, 3, 4, 5];
//La función parámetro generalmente es una función flecha sin cuerpo.
const encontrado = numeros.find(elemento => elemento > 3); //Encuentra 4

const nombres     = ["Ana", "Ema", "Juan"];
const encontrado2 = nombres.find(elemento => elemento === "Ema"); //Encuentra "Ema"
const encontrado3 = nombres.find(elemento => elemento === "Alan");//undefined
```

El parámetro de estos métodos es una función, y generalmente es una flecha sin cuerpo, porque se usa un único criterio para realizar la búsqueda o el filtro.

Filter

El método ***filter()*** crea un nuevo array con todos los elementos que cumplan la función de comprobación enviada por parámetro. Generalmente se obtiene un array con menos elementos que la lista a filtrar:

```
const numeros = [1, 2, 3, 4, 5];
const filtro1 = numeros.filter(elemento => elemento > 3); //Encuentra [4,5]
const filtro2 = numeros.filter(elemento => elemento < 4); //Encuentra [1,2,3]

const nombres = ["Ana", "Ema", "Juan", "Elia"];
//Filtrar nombre que incluyen la letra "n" Encuentra ["Ana","Juan"]
const filtro3 = nombres.filter(elemento => elemento.includes("n"));
```

Map

El método ***map()*** se utiliza para crear un nuevo array con todos los elementos del array original, transformados según las operaciones de la función enviada por parámetro. El

nuevo array obtenido tiene la misma cantidad de elementos que el array original, pero con los valores modificados:

```
const numeros  = [1, 2, 3, 4, 5];
const porDos   = numeros.map(x => x * 2);   //porDos = [2, 4, 6, 8, 10]
const masCien  = numeros.map(x => x + 100); //porDos = [102, 104, 106, 108, 110]

const nombres = ["Ana", "Ema", "Juan", "Elia"];
const lengths = nombres.map(nombre => nombre.length);//lengths = [3, 3, 4, 4]
```

Veamos a continuación un ejemplo aplicado, donde se utilizan los métodos presentados en una solución más aproximada al escenario de la tienda virtual.

```
const productos = [{ id: 1,  producto: "Arroz", precio: 125 },
                   {  id: 2,  producto: "Fideo", precio: 70 },
                   {  id: 3,  producto: "Pan"  , precio: 50},
                   {  id: 4,  producto: "Flan" , precio: 100}];

const buscarPan = productos.find(producto => producto.id === 3);
console.log(buscarPan);//{id: 3, producto: "Pan", precio: 50}

const baratos = productos.filter(producto => producto.precio < 100);
console.log(baratos);//
[{id: 2,producto:"Fideo",precio:70},{id:3,producto:"Pan",precio: 50}]

const aumentos = productos.map(producto => producto.precio += 30);
console.log(aumentos);
//[155, 100, 80, 130] y el valor de cada producto cambio.
```