

Arrays y objetos (Parte I)

Estructura y utilidad de un Array

Competencias

- Reconocer la estructura y utilidad de un arreglo para crear, acceder y realizar adecuadamente operaciones con sus elementos.
- Desarrollar un programa utilizando operaciones de creación y acceso a los elementos de un arreglo acorde al lenguaje JavaScript para resolver un problema.

Introducción

A medida que comenzamos a desarrollar aplicaciones, en ocasiones veremos la necesidad de agrupar los datos que generamos dentro de nuestra aplicación para mantener cierto orden y control. Para apoyarnos con esta tarea nos encontramos con los Arrays que nos permite estructurar nuestra información en arreglos de una dimensión y/o bidimensionales llamados matrices.

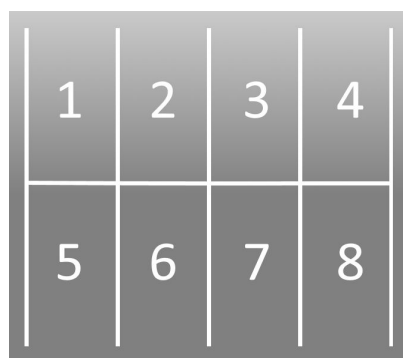
Este tipo de estructuras son clave para trabajar con colecciones de datos y comprender su sintaxis, nos permite manejar estructuras más complejas, como los objetos.

A continuación, revisaremos en detalle qué son los arreglos, qué herramientas nos provee JavaScript para trabajar con ellos y cómo utilizamos este tipo de estructuras en problemas cotidianos.

¿Qué es un Array?

Un array o arreglo en JavaScript es un tipo de dato que nos permite almacenar de manera ordenada un conjunto de elementos, a los que podemos acceder desde una sola variable. Este tipo de datos es útil cuando queremos guardar una lista de elementos, por ejemplo, un listado de nombres, una lista de compras, etc.

Para hacernos una idea mejor del concepto de arrays o arreglos, pensemos en los estacionamientos de un centro comercial, como se muestra en la siguiente imagen:



1	2	3	4
5	6	7	8

Imagen 1. Ejemplo de un array basado en una analogía de un estacionamiento.

Fuente: Desafío Latam

Los estacionamientos son al igual que los arrays, un espacio asignado para agrupar o almacenar ciertos tipos de elementos. En el caso de los estacionamientos, están destinados para agrupar objetos tales como automóviles, motocicletas, camiones, etc. Los arrays por su parte, tienen el objetivo de almacenar diversos tipos de elementos con distintos valores y para ciertos tipos de datos (números, cadenas de textos, valores booleanos, decimales, objetos, entre otros.).

Declaración y asignación de Arrays

Para definir arrays en JavaScript puedes utilizar los editores de código [JSFiddle](#), [CodePen](#), [CodeSandbox](#) o un archivo HTML con código JavaScript en su interior.

Los arrays en JavaScript son comúnmente definidos entre corchetes ("[]"), indicando en su interior los valores que necesitamos listar, aunque también existen otras formas de hacerlo. Por ejemplo, si quisiera agrupar una lista de amigos dentro de un array, podría escribir y ejecutar el siguiente código:

```
let amigos = ["Erick", "Cristian", "Max", "Claudia"];  
console.log(amigos);
```

Como podemos ver en la primera línea del código, creamos una variable llamada **amigos** a la cual le asignamos (dentro de corchetes y separados por una coma) los valores **Erick, Cristian, Max y Claudia**. En la segunda línea, utilizamos el **console.log** al cual le pasamos como parámetro el array creado. Al ejecutarse este código, se desplegará un mensaje desde la consola del navegador con la lista de amigos, como se muestra a continuación:

```
Array(4) [ "Erick", "Cristian", "Max", "Claudia" ]
```

Ejercicio guiado: Lista de películas

Crear una lista de películas utilizando un array, las películas a agregar son: "It 2, Rambo 3, Halloween, Shaft", sigamos los siguientes pasos:

- **Paso 1:** Crear e inicializar una variable para que contenga nuestra lista de películas.
- **Paso 2:** Dentro de la notación de corchetes y separadas por comas, se escribe cada película de la lista como string, es decir, dentro de las comillas dobles o sencillas.

```
let peliculas = ["It 2", "Rambo 3", "Halloween", "Shaft"];  
console.log(peliculas);
```

- **Paso 3:** Ejecutar el código en la consola del navegador web, obteniendo el siguiente resultado:

```
Array(4) [ "It 2", "Rambo 3", "Halloween", "Shaft" ]
```

Como puedes darte cuenta, no sólo es posible crear arrays con la notación anterior, sino que también, podemos definirlos a través del objeto `Array()`. Para la lista de películas recién creada, podemos utilizar el objeto **Array** de la siguiente forma.

```
let peliculas = new Array("It 2", "Rambo 3", "Halloween", "Shaft");
```

O de esta manera:

```
var peliculas = Array("It 2", "Rambo 3", "Halloween", "Shaft");
```

En resumen, existen diversas maneras de almacenar datos del tipo array en una variable, por lo que cualquiera de las tres formas presentadas son válidas y utilizadas, pero comúnmente

encontrarás la definición de variables con datos del tipo array como lo indica la primera opción, ejemplo: `var frutas = ["pera", "manzana", "membrillo", "uvas"];`

Arrays vacíos

Existirán casos en donde necesitarás inicializar un array vacío, ya que le asignaremos valores en otro momento. Estos Arrays son dinámicos o dependen de alguna otra rutina, de ser así, se podría asignar un array vacío a la variable películas de la siguiente forma:

```
var peliculas = [];
```

Por si te lo preguntas, al intentar visualizar el resultado de un array vacío dentro de un alert, se mostrará un mensaje en blanco en el navegador:

```
alert(peliculas);
```

Arrays con distintos Tipos de Datos

Los arrays no tan sólo aceptan cadenas de textos o strings como revisamos en los casos anteriores, sino que también admiten otros tipos de datos y objetos.

Podemos realizar los siguientes códigos para cada uno de los ejemplos:

- Para crear una lista de valores numéricos:

```
var pares = [2, 4, 6, 8, 10];
```

- Para crear una lista de valores decimales:

```
var decimales = [2.1, 4.4, 3.1, 7.7];
```

- Para crear una lista de valores booleanos (verdadero o falso):

```
var booleanos = [false, false, true];
```

- Para crear una lista de objetos dentro de un arreglo con una lista de nombres:

```
var objetos = [{nombre: 'juan'}, {nombre: 'carlos'}];
```

Como podemos ver los objetos se definen dentro de llaves ("{ } "), en los cuales en su interior se puede ver primero el nombre de una propiedad junto con su valor. En este último caso la **propiedad** para cada objeto es **nombre** y el **valor** es **juan** y **carlos** respectivamente. De todos modos, abordaremos de manera más detenida el uso de objetos en los siguientes capítulos.

Ejercicio guiado: Arrays con distintos tipos de datos

Crear un array que contenga distintos tipos de datos, en el espacio 0 debe ir almacenado tu primer nombre, en el siguiente espacio tu primer apellido, en el siguiente espacio las variables edad y fecha de nacimiento como objeto, y para finalizar indicar si eres hijo único mediante operadores booleanos. Para ello:

- **Paso 1:** Declarar la variable que contendrá el arreglo con los corchetes pero vacía, quedando:

```
var datosPersonales = [,,{,},];
```

- **Paso 2:** Creado y distribuido el espacio en el arreglo, ahora agregamos los datos correspondientes:

```
var datosPersonales = ['juan', 'duran', {edad:34, nacimiento:1985}, false];  
console.log(datosPersonales);
```

- **Paso 3:** Al ejecutar el código el resultado sería:

```
0: "juan"  
1: "duran"  
2: {edad: 34, nacimiento: 1985}  
3: false
```

Ejercicio propuesto (1)

Crear una estructura que contenga distintos tipos de datos, crea una variable del tipo array que contenga los siguientes datos: "Perro", "Gato", ave: "guacamaya", 2020, "erizo", lugar: "Chile", ciudad: "Santiago de Chile". Luego muestra el resultado en la consola del navegador.

Referenciando Arrays

Ahora que hemos aprendido a crear arreglos, veremos cómo trabajar con los datos que contiene. Para acceder a un valor específico de un array, podemos valernos de su **índice asignado** para realizar dicha tarea. Por ejemplo, para acceder al segundo elemento del array de la variable **películas** o al valor **Rambo 3**, podemos realizarlo de la siguiente manera:

```
var peliculas = [  
  "It 2", // posición 0  
  "Rambo 3", // posición 1  
  "Halloween", // posición 2  
  "Shaft" // posición 3  
];  
console.log(peliculas[1]);
```

Al ejecutar el código anterior, el resultado de traer la posición 1 del array de películas es:

```
"Rambo 3"
```

Como se puede apreciar, dentro del `console.log` escribimos el nombre de la variable **películas** junto a corchetes que contienen el **índice** que apunta al valor **Rambo 3**. Considera que los índices parten desde el **0** hasta el número total de elementos menos 1. En este caso, el elemento con índice **0** sería **It 2**, así como el último (**Shaft**) corresponde al índice **3**.

Ejercicio guiado: Accediendo a los valores de un arreglo

Acceder y mostrar los datos con el nombre de: "Vue JS" y Backend: "Node JS", del siguiente arreglo:

```
var programa_web = [  
  "JavaScript",  
  "React JS",  
  "Vue JS",  
  "Angular JS",  
  {Backend: "Node JS"}  
];
```

¿Lograste acceder y mostrar los valores indicados? Ahora lo vamos a hacer en conjunto, para ello, recuerda que los arreglos tienen posiciones definidas desde el cero en adelante, por ende:

- **Paso 1:** Para acceder a la posición que contiene el valor de "Vue JS", debemos referenciar la variable que contiene el arreglo con el número dos (2).
- **Paso 2:** Para acceder a la variable "BackEnd" con el valor de "Node JS", se debe referenciar la variable programa_web, que contiene el arreglo, con la posición 4. Quedando de la siguiente manera:

```
var programa_web = [  
  "JavaScript", // posición 0  
  "React JS", // posición 1  
  "Vue JS", // posición 2  
  "Angular JS", // posición 3  
  {BackEnd: "Node JS"} // posición 4  
];  
  
console.log(programa_web[2]);  
console.log(programa_web[4]);
```

- **Paso 3:** Al ejecutar en el navegador web el código anterior, obtendremos el siguiente resultado:

```
"Vue JS"  
{BackEnd: "Node JS"}
```

Arreglos multidimensionales o matrices

Una matriz es un conjunto ordenado de elementos (igual que un arreglo) en una estructura de filas y columnas, donde cada combinación (fila o columna) contiene un elemento o valor:

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}$$

Imagen 2. Matriz de nxn

Fuente: [Wikipedia](https://es.wikipedia.org/wiki/Matriz)

Nativamente JavaScript no provee arrays multidimensionales (matrices) como otros lenguajes de programación, pero puedes crear una matriz definiendo un array en donde cada elemento será uno de ellos:

```
var array = [  
  "elemento1",  
  "elemento2",  
  "elemento3"  
]
```

En el ejemplo anterior, definimos un array como lo hemos hecho a lo largo de la lectura, pero ahora lo transformaremos en una matriz reemplazando cada elemento por un array:

```
var matriz = [  
  ["elemento1"],  
  ["elemento2"],  
  ["elemento3"]  
];  
console.log(matriz[0])  
console.log(matriz[1])  
console.log(matriz[2])
```

Resultado:

```
(0) ["elemento1"]  
(1) ["elemento2"]  
(2) ["elemento3"]
```

Ejercicio guiado: Matrices

Realizar una matriz de 3x3 con números enteros desde el 1 hasta el 9, tres números por cada arreglo interno. En este caso, cada espacio del arreglo original contendrá otro arreglo dentro de él con los valores de la matriz, es decir, tendremos arreglos anidados dentro de la variable que ya es un arreglo. Resolvamos este ejemplo:

- **Paso 1:** Construir el arreglo principal y lo llamaremos matriz3x3, dejando planteados los tres arreglos internos:


```
var matriz3x3 = [  
  [elementos1],  
  [elementos2],  
  [elementos3],  
]
```

- **Paso 2:** Al plantear el arreglo principal con los internos, se agregan los números correspondientes a cada arreglo interno, es decir, tres números por arreglos para poder formar la matriz de 3x3. El primer elemento lo compone los números del 1 al 3, para el segundo elemento los números del 4 al 6 y para el tercer y último elemento los números del 7 al 9.

```
var matriz3x3 = [  
  [1,2,3],  
  [4,5,6],  
  [7,8,9],  
];
```

- **Paso 3:** Mostramos el arreglo realizado en la consola del navegador web mediante un console.log:

```
console.log(matriz3x3);
```

- **Paso 4:** Al ejecutar y mostrar el código anterior, el resultado sería:

```
0: Array(3) [ 1, 2, 3 ]  
1: Array(3) [ 4, 5, 6 ]  
2: Array(3) [ 7, 8, 9 ]
```

Si llevamos esto a una representación más legible, como en el caso de una tabla, podremos observar dónde (posición y lugar) se encuentra cada valor de la matriz:

matriz3x3[i]	0	1	2
matriz3x3[0]	1	2	3
matriz3x3[1]	4	5	6
matriz3x3[2]	7	8	9

Tabla 1. Distribución de la variable matriz3x3 en una tabla.

Fuente: Desafío Latam

Entonces, si queremos acceder al número 6 deberíamos hacerlo acercándonos a la posición `matriz3x3[1][2]`, donde el valor 1 representa a la fila y el 2 a la columna. Por ejemplo, si queremos acceder al valor número siete (7), utilizamos el nombre de la variable más los corchetes para indicar cuál sería el primer espacio del arreglo (fila) y cuál el espacio en el arreglo interno que contiene los elementos (columnas), quedando de la siguiente manera:

```
console.log(matriz3x3[2][0])
```

Y el resultado sería:

```
7
```

Ejercicio propuesto (2)

Escribir las coordenadas del array que muestran los valores 1, 13 y 34 de la siguiente matriz:

```
var matriz3x3 = [  
  ["1", "2", "3"],  
  ["5", "8", "13"],  
  ["21", "34", "55"],  
];
```

Largo de Arrays y su manipulación

Los métodos para manipular matrices son los mismos que para manipular un array, la diferencia es que en vez de estar trabajando con elementos u objetos, tenemos un arreglo. Por consiguiente, de manera implícita, los arrays contienen una propiedad llamada **length**, la cual permite acceder a la cantidad de elementos que contienen. Si quisiéramos saber la cantidad de valores que posee la variable **películas** que creamos anteriormente, podemos referenciarla seguida de un **punto** junto con el nombre de la propiedad (**length**):

```
var peliculas = ["It 2", "Rambo 3", "Halloween", "Shaft"];  
console.log(peliculas.length);
```

El resultado de lo anterior será en el "console.log" con el valor **4**, dado que nuestra variable **películas** contiene 4 valores. Así mismo, se debe tener cuidado con el uso de la propiedad **length**, ya que de manera explícita (o intencional) puede contener un valor distinto al esperado, dado que su valor podemos reasignarlo, lo que conlleva incluso a eliminar o

expandir un array. Por ejemplo, podemos cambiar el valor de **length** de la variable **peliculas** de **4** a **3** de la siguiente forma:

```
var peliculas = ["It 2", "Rambo 3", "Halloween", "Shaft"];
peliculas.length = 3;
console.log(peliculas.length);
```

Con esto, el `console.log` imprimirá el valor 3, ahora cada vez que se modifiquen los valores del array, el valor de **length** volverá a reflejar la cantidad de elementos que posee. Ahora, ¿Qué sucede en el caso anterior si quisiera acceder al valor **Shaft**? Si se reduce como se hizo anteriormente el arreglo de 4 elementos a 3 utilizando la instrucción `length`:

```
var peliculas = ["It 2", "Rambo 3", "Halloween", "Shaft"];
peliculas.length = 3;
console.log(peliculas[3]);
```

El resultado de lo anterior será **undefined**, esto nos dice que el valor con índice 3 no existe o no tiene un valor definido. Ésto se debe a que en el momento de modificar el valor de la propiedad **length**, también modificamos la capacidad de nuestro arreglo, lo que a su vez provocó que el elemento **Shaft** se elimine. Ahora, si en vez de haber reducido el valor de **length** lo hubiéramos expandido, ningún elemento se habría eliminado.

Finalmente, a un array se le puede asignar un valor con un índice predefinido por nosotros, sin importar si quedan espacios sin definir o rellenar con otro elemento:

```
var peliculas = ["It 2", "Rambo 3", "Halloween", "Shaft"];
peliculas[5] = "Inferno";
console.log(peliculas.length);
```

En este caso, agregaremos una nueva película a la lista llamada Inferno con el índice 5 y desplegamos el largo por pantalla a través de un `console.log()`. Esta modificación provoca que el array pase de tener un largo de 4 a 6, dejando el espacio cinco (5) como un elemento vacío sin ningún tipo de dato:

```
["It 2", "Rambo 3", "Halloween", "Shaft", , "Inferno"]
```

Ejercicio guiado: Lista de instrumentos musicales

Crear un nuevo arreglo con una lista de instrumentos musicales: "Guitarra Eléctrica, Piano de Cola, Violín, Trompeta". En base a esta colección, modificar la cantidad de elementos disponibles, su largo y agregaremos nuevos elementos al arreglo ("Guitarra Clasica, Chelo") en las posiciones 6 y 7 respectivamente.

- **Paso 1:** Crear la estructura básica del arreglo, es decir, inicializar la variable y dejar los corchetes para agregar los elementos dentro del arreglo.

```
let instrumentos = [];
```

- **Paso 2:** Es momento de agregar los elementos dentro del arreglo planteados en el enunciado, es decir: "Guitarra Eléctrica, Piano de Cola, Violín, Trompeta", por lo que tienen que ir dentro de los corchetes con comillas dobles o simples para dejarlos como string y separados por comas. Luego, mostramos la cantidad de elementos dentro del arreglo mediante la instrucción `length` en la consola del navegador:

```
let instrumentos = ["Guitarra Eléctrica", "Piano de  
cola", "Violín", "Trompeta",];  
console.log(instrumentos);  
console.log(instrumentos.length);
```

- **Paso 3:** Al ejecutar el código anterior, el resultado en la consola del navegador sería:

```
Array(4) [ "Guitarra Eléctrica", "Piano de cola", "Violín", "Trompeta" ]  
4
```

- **Paso 4:** Para poder modificar el largo original del arreglo, podemos indicarle mediante la propiedad `length`, la nueva cantidad de elementos totales del arreglo y mostramos en la consola el arreglo para poder visualizar la modificación realizada:

```
instrumentos.length = 2;  
console.log(instrumentos);  
console.log(instrumentos.length);
```

- **Paso 5:** Al ejecutar el código anterior, el resultado en la consola del navegador sería:

```
Array [ "Guitarra Eléctrica", "Piano de cola" ]  
2
```

- **Paso 6:** Ahora para agregar los dos nuevos elementos que se solicitan en el enunciado, utilizamos la variable del arreglo y le indicamos en cual posición se desea agregar cada elemento. La posición se indica con corchetes y el número donde debían ir los elementos:

```
instrumentos[6] = "Guitarra Clásica";  
instrumentos[7] = "Chelo";  
console.log(instrumentos);  
console.log(instrumentos.length);
```

- **Paso 7:** Al ejecutar el código anterior, el resultado en la consola del navegador sería:

```
Array(8) [ "Guitarra Eléctrica", "Piano de cola",,,, "Guitarra  
Clásica", "Chelo" ]  
8
```

Ejercicio propuesto (3)

Para el arreglo que se presenta a continuación, mostrar por la consola del navegador web la cantidad de elementos totales que contiene el arreglo. Luego, reduce en un elemento el arreglo actual, es decir, que pase de 5 elementos a 4, y por último, añade un nuevo elemento en la posición 7 del arreglo denominado: "Express".

```
let datosWeb = ["JavaScript", "VueJS", "AngularJS", "ReactJS", "NodeJS"]
```

Iterando sobre arreglos

Competencia

- Realizar operaciones iterativas sobre elementos de un arreglo utilizando estos recursos para resolver un problema.

Introducción

Entender cómo trabajar con arreglos en JavaScript y en la mayoría de los lenguajes de programación, es importante además de útil para el trabajo cotidiano.

Una vez que dominamos la creación de estas estructuras, cómo recuperar y agregar elementos, es importante entender qué métodos nos provee el lenguaje para recorrerlos utilizando iteradores.

A continuación, veremos en detalle estos conceptos, que nos permitirán optimizar la resolución de aquellos problemas que requieran trabajar con ciclos y colecciones de datos.

Recorriendo los valores de un Array

Una de las tareas más usuales dentro de la programación en JavaScript es recorrer arrays o colecciones de datos, y poder manipular cada uno de sus elementos. Para esta tarea, tenemos a disposición rutinas que nos permiten iterar, como las siguientes:

ciclo for

```
var peliculas = ["It 2", "Rambo 3", "Halloween", "Shaft"];
for (var i = 0; i < peliculas.length; i++) {
    document.write(peliculas[i] + '|');
};
```

Para recorrer cada una de las películas dentro del arreglo anterior, primero escribimos la palabra for, la cual entre paréntesis se compone de 3 bloques separados por punto y coma:

- **Definición de una variable** que nos servirá como índice. Generalmente se establece en 0, dado que los índices de arreglos y colecciones parten con este valor.
- **Evaluación de la variable** creada anteriormente. Si la evaluación resulta positiva, se ejecuta la rutina al interior del ciclo for, de lo contrario, se termina la ejecución del ciclo y el código continúa con la ejecución en el siguiente bloque. En el caso anterior, se evalúa si el valor de la variable i es menor al largo de elementos del array.
- **Modificación de la variable** creada al inicio (i). Nos sirve para iterar sobre un elemento distinto. En el caso anterior, se hace un incremento de 1 después de que la evaluación resulta positiva y se ejecuta la rutina al interior de las llaves ({ }).

Con la evaluación positiva, lo que hacemos es hacer uso de la función **document.write**, la cual nos permite escribir en una página lo que se encuentre dentro de sus paréntesis (parámetros). En el ejemplo anterior, se accede a cada valor del arreglo películas en función del valor que tenga la variable i, la cual muta cada vez que se cumple con la evaluación.

La ejecución del código ya visto en el ciclo for nos entregará el siguiente resultado:

```
It 2|Rambo 3|Halloween|Shaft
```

Sin embargo, recorrer arrays mediante el **ciclo for** no es la única forma de iterar sobre éstos. En JavaScript, todos los arrays heredan por defecto algunas funciones que nos ayudarán a realizar iteraciones o bucles sobre estos elementos de diferentes maneras y con diferentes resultados, las cuales revisaremos más adelante.

Ejercicio guiado: Recorriendo un arreglo con un ciclo for

Mostrar cada uno de los elementos que componen el arreglo indicado, utilizando la estructura repetitiva “for” según el siguiente código:

```
var usuarios = [  
  'Manuel',  
  {nombre: 'Juan', edad: 34},  
  30,  
  {nombre: 'Rodrigo', edad: 55},  
  true,  
  {nombre: 'Paola', edad: 34},  
  'Pedro',  
  ['Maria', 'Eliana']  
];
```

- **Paso 1:** Crear una carpeta en tu lugar de trabajo favorito y dentro de ella crea dos archivos, un index.html y un script.js.
- **Paso 2:** En el archivo script.js, se debe armar la estructura repetitiva o ciclo for, el cual, servirá para recorrer el arreglo en cada uno de sus espacios y mostrar el contenido que esté disponible, por lo tanto, copiamos la variable con el arreglo del enunciado y luego se crea la estructura del ciclo for implementando la instrucción “length” para tener disponible la longitud del arreglo e iterar en base a ese número:

```
var usuarios = [  
  'Manuel',  
  {nombre: 'Juan', edad: 34},  
  30,  
  {nombre: 'Rodrigo', edad: 55},  
  true,  
  {nombre: 'Paola', edad: 34},  
  'Pedro',  
  ['Maria', 'Eliana']  
];  
  
for (var i = 0; i < usuarios.length; i++) {  
};
```


- **Paso 3:** Dentro de la estructura del ciclo for, se mostrará cada uno de los elementos encontrados, accediendo mediante la referencia dispuesta por la variable del ciclo for que lleva el conteo de las iteraciones, en este caso "i".

```
for (var i = 0; i < usuarios.length; i++) {  
  console.log(usuarios[i]);  
};
```

- **Paso 4:** Ejecutamos el código anterior en nuestro navegador mediante el archivo index.html y entramos a la consola para poder observar el resultado, el cual sería:



Imagen 3. Resultado de la función en el navegador.
Fuente: Desafío Latam

Como se puede observar en la imagen anterior, al recorrer un arreglo con el ciclo for se tiene acceso a cada uno de los elementos dentro del mismo, sin importar si es un arreglo o un objeto. Posteriormente se estudiarán otros métodos y técnicas para acceder a los objetos dentro de un arreglo.

Ejercicio propuesto (4)

En base al siguiente código, escribe un ciclo que acceda y muestre con un console.log cada uno de los elementos dispuestos en el siguiente arreglo mediante el ciclo repetitivo for:

```
var mascotas = ['Perros', {nombre: 'Firulais', edad: 5}, 'Gatos', {nombre: 'Michi', edad: 2}, 'Aves', {nombre: 'Pepito', edad: 1},]
```

for...in

La estructura **for...in** es utilizada para realizar ciclos “finitos” sobre objetos (Object), es decir, se recorren de inicio a fin sin necesidad de un inicializador y no existe una condición de salida, sino que es implícita y corresponde al final de los elementos iterables.

```
for (variable in objeto) {  
  // ...  
}
```

Ejercicio guiado: Recorriendo objetos con for... in

Se solicita recorrer, acceder y mostrar cada una de las llaves (variables) para un objeto que represente a una persona con nombre, edad y una función de saludo, este objeto es:

```
let persona = {  
  nombre: 'Marcelo Salas',  
  edad: 11,  
  saludar: () => {  
    return "Buena matador!";  
  }  
};
```

- **Paso 1:** Crear una carpeta en tu lugar de trabajo favorito y dentro de ella crea dos archivos, un index.html y un script.js.
- **Paso 2:** En el archivo script.js, se debe armar la estructura for...in, la cual, servirá para recorrer el objeto en cuestión, accediendo a la llave y mostrando en la consola del navegador cada propiedad o llave que tenga el objeto, más no el valor que acompaña a cada propiedad, por consiguiente, el código del for...in quedaría:

```
let persona = {  
  nombre: 'Marcelo Salas',  
  edad: 11,  
  saludar: () => {  
    return "Buena matador!";  
  }  
}
```

```
for (propiedad in persona) {  
  console.log(propiedad);  
}
```

- **Paso 3:** Ejecutamos el código en nuestro navegador web e ingresamos a la consola del navegador y encontraremos el resultado:

```
nombre  
edad  
saludar
```

Es posible entonces afirmar que los elementos iterables de un objeto corresponden a las claves/índices/propiedades de este y que **for...in** asigna de manera automática estos valores sin necesidad tampoco de hacer explícita la condición de término. Por ende, es necesario recalcar que no es posible “recorrer” o iterar sobre los elementos de un objeto de manera convencional haciendo uso de `for(;;)` y `while/do...while`.

for...of

Ya conocimos cómo iterar sobre objetos, de igual manera también es posible hacer ciclos sobre “objetos iterables”, el más conocido y ya visto anteriormente en este curso son los Array. La estructura de **for...of** corresponde a:

```
for (variable of objeto) {  
  // ...  
}
```

Como ejemplo de uso del ciclo `for...of`, podríamos ver el siguiente arreglo y la aplicación directa de la estructura `for...of` al arreglo, siendo la siguiente:

```
let data = ['casa', '12', true, 'JS']  
for (let values of data) {  
  console.log(values);  
};
```

Obteniendo como respuesta al ejecutar el código:

```
casa
12
true
JS
```

Ejercicio guiado: Recorriendo arreglos con for...of

A partir de un array con las edades de los miembros de una familia de 5 personas, se debe recorrer y mostrar los datos por consola, utilizando tanto for...in como for...of. El arreglo con las edades es el siguiente:

```
let edades = [49, 51, 21, 18, 15];
```

- **Paso 1:** Crear una carpeta en tu lugar de trabajo favorito y dentro de ella crea dos archivos, un index.html y un script.js.
- **Paso 2:** Ahora se debe armar primeramente la estructura for...of, la cual, servirá para recorrer el arreglo en cuestión, accediendo a los valores del arreglo y mostrando el resultado en la consola del navegador, por consiguiente, el código del for...of quedaría:

```
let edades = [49, 51, 21, 18, 15];

for (let edad of edades) {
  console.log(edad);
}
```

- **Paso 3:** Ejecuta el código en el navegador web, por lo que el resultado sería:

```
49
51
21
18
15
```

- **Paso 4:** Ahora como complemento al ejercicio anterior, recorramos el mismo arreglo pero esta vez utilizando **for..in**, por lo que agregaremos esta última estructura dejando el for...of anterior para comparar ambos resultados en línea:

```
let edades = [49, 51, 21, 18, 15];

for (let otra of edades) {
  console.log(otra);  //
}

for (let otra in edades) {
  console.log(otra);  //
}
```

- **Paso 5:** Al ejecutar el código anterior, el resultado mostrado en la consola del navegador quedaría:

```
49
51
21
18
15
0
1
2
3
4
```

Por consiguiente, la diferencia entre ambos es que el **for...in** itera sobre los índices (nombres de las propiedades), mientras que el **for...of** itera sobre los valores de las propiedades.

Para comprender mejor cómo trabajar con arreglos, es importante entender en qué consiste el álgebra de conjuntos. Puede parecer inicialmente un tema alejado de lo que estamos abordando en la lectura, pero te darás cuenta que tienen mucho en común con las acciones que realizamos con los arrays y objetos. Para esto, puedes consultar el documento **Material Apoyo Lectura - Álgebra de conjuntos**, ubicado en “Material Complementario”.

Ejercicio propuesto (5)

Accede y muestra mediante un `console.log` cada uno de los elementos dispuestos en el siguiente arreglo mediante los ciclos repetitivos *for...in* y *for...of*:

```
let mascotas = ['Perros', {nombre: 'Firulais', edad: 5}, 'Gatos', {nombre: 'Michi', edad: 2}, 'Aves', {nombre: 'Pepito', edad: 1},]
```

Estructura y Utilidad de un Objeto

Competencias

- Reconocer la estructura y utilidad de un objeto para crear, acceder y realizar adecuadamente operaciones con sus elementos.
- Codificar un programa utilizando objetos para resolver un problema acorde al lenguaje JavaScript ES6.

Introducción

A primera vista, puede parecer suficiente disponer de tipos de datos de distinta clase, como numéricos, booleanos, decimales, strings, etc., para desarrollar el código que necesitamos. Sin embargo, en ocasiones vamos a requerir algo mucho más complejo que sólo estos mecanismos, como por ejemplo, definir variables o elementos en nuestro código que sean capaz de interpretar valores más complejos. Para esto, disponemos de objetos en JavaScript (y en la gran mayoría de lenguajes de programación), los cuales veremos a continuación.

El uso de objetos es el punto de partida de estructuras más complejas, manejar este concepto tanto teórico como prácticamente, nos facilitará resolver problemas que requieran trabajar con colecciones de datos de distinto tipo, lo cual es parte de las tareas cotidianas de cualquier desarrollador.

¿Qué es un Objeto?

Los objetos en JavaScript en palabras sencillas, son uno o varios conjuntos de propiedades y/o métodos que están representados mediante la definición de una variable. Cada una de estas propiedades o métodos, obedece a una relación **nombre, valor o clave**, donde el nombre o clave es un puntero o referencia al valor asignado.

Podemos pensar en los objetos como contenedores que pueden mantener valores diferentes y estar relacionados entre sí. Por ejemplo, los usuarios de un sitio web, las facturas de ventas de una tienda de ventas, el inventario de suministros de un hospital, etc. Pueden ser objetos en JavaScript.



Imagen 4. Ejemplificación de un objeto, junto con sus propiedades.
Fuente: Desafío Latam

En la imagen 4, podemos ver la representación de un objeto llamado **Automóvil**, este objeto tiene algunas propiedades listadas, las cuales son **Marca, Modelo y Origen**. Cada propiedad tiene asociada un valor que describe dicho objeto de manera particular y simplificada. Esta es la idea detrás de los objetos, describir elementos del mundo real (o no), junto con las características que nos interesan rescatar del mismo, para los fines que estimemos conveniente dentro de nuestros desarrollos.

Una de las ventajas de los objetos, es que dentro de ellos podemos definir elementos con distintos tipos de datos en un mismo elemento, conteniendo por ejemplo, cadenas de texto, booleanos, decimales, etc.

Podemos pensar en los objetos como **el resultado de una idea simple** tomada desde un concepto que puede ser mucho más complejo. Esto se conoce como abstracción y es uno de los pilares de la programación orientada a objetos.

Declaración e Inicialización de un Objeto

Un objeto puede ser instanciado definiendo una variable y asignándole los valores requeridos entre llaves, los cuales pueden estar presentes o no en un comienzo (tal cual como se hace con los arreglos):

```
var automovil = {};
```

Si tratamos de imprimir el objeto **automóvil** que acabamos de crear mediante la función **alert**, obtendremos la siguiente salida:

```
[objeto Objeto]
```

Esto quiere decir que acabamos de instanciar nuestro primer objeto. Esta forma de declaración e inicialización de objetos da como resultado un **objeto literal**. Los objetos literales son ampliamente utilizados hoy en día, debido a la manera sencilla de escribirlos y transferirlos, por ejemplo, a una base de datos o un servicio web. No sólo este tipo de objetos son más fáciles de transferir, sino que también de procesar ya que se tratan como si fuesen arreglos.

Ahora, este objeto por sí solo no nos dice mucho y tiene poca utilidad de momento, por lo que le agregaremos algunos elementos. Para ésto, modificaremos el código anterior:

```
var automovil = {  
  marca: 'Mazda',  
  modelo: '3 sport',  
  patente: 'LJKH63',  
  color: 'azul',  
  kilometraje: 15000,  
  usado: false,  
  encender: function(){  
    alert('automóvil encendido.');  }  
};
```

Ahora nuestro objeto **automóvil** está mucho más completo y podemos notar alguna de las cosas que mencionamos anteriormente: el objeto contiene propiedades y métodos en su interior, los cuales obedecen a una estructura de **nombre y valor**. Por ejemplo, la propiedad con nombre marca contiene el valor **Mazda**. También podemos ver que tenemos propiedades que contienen valores de tipo cadenas de texto (strings), un booleano, un numérico e incluso una función (encender).

Acceso a las propiedades de un Objeto

Para acceder a las propiedades de un objeto, podemos realizarlo de 2 maneras distintas: mediante la notación de punto o la notación de corchetes. Ambas notaciones son válidas, aunque la notación de puntos es la que comúnmente es más vista puesto su simplicidad.

Volveremos al objeto **automóvil** que creamos anteriormente y trataremos de acceder a la propiedad patente. Esto se puede hacer de la siguiente forma con la notación de puntos:

```
automovil.patente
```

O también, se puede hacer así con la notación de corchetes, que opera de la misma manera como vimos anteriormente con los arrays.:

```
automovil['patente']
```

Si imprimimos cualquiera de los códigos anteriores mediante la función **console.log**, obtendremos el siguiente resultado:

```
LJKH63
```

Ejercicio guiado: Accediendo a las propiedades de un objeto

Se solicita mostrar los valores de raza, peso y amigable existentes en el objeto, implementando las notaciones de acceso:

```
let perro = {  
  raza: 'Pastor Alemán',  
  origen: 'Alemania',  
  pelaje: 'Lanudo',  
  peso: '33kg',  
  edad: 12,  
  amigable: true,  
  sonidos: function(){  
    console.log('El perro ladra');  
  }  
};
```

- **Paso 1:** Ahora, para acceder a esos valores solicitados en el enunciado del ejercicio, debemos implementar cualquiera de las dos notaciones mostradas al inicio. Con la primera notación se utiliza el punto para separar la variable global que contiene el objeto de los elementos o llaves que queremos acceder para mostrar el valor que contengan, por lo que quedaría:

```
console.log(perro.raza);  
console.log(perro.peso);  
console.log(perro.amigable);
```

- **Paso 2:** Agregar sobre el mismo código el acceso a los valores del objeto mediante la notación con corchetes, quedando:

```
console.log(perro['raza']);  
console.log(perro['peso']);  
console.log(perro['amigable']);
```

- **Paso 3:** Ejecutar el código anterior y obtendrás el mismo resultado en ambos casos, por lo que es igual utilizar una notación u otra:

```
Pastor Alemán  
33kg  
true
```

Comúnmente, el primer método donde se emplea el punto (.) es el más utilizado para acceder a los objetos y es el que usaremos a lo largo del curso para todos los ejemplos.

Acceso a las funciones de un Objeto

El acceso a las funciones es similar al cómo se realizan las propiedades de un objeto, con una salvedad que es necesario aclarar, puesto que se puede caer en un error muy común a la hora de desarrollar.

Si intentamos acceder a la función **encender** del objeto **automovil** de la misma forma que lo hacemos con alguna propiedad y pretendemos ejecutarlo, veremos que tendremos un resultado distinto al esperado. Por ejemplo, si ejecutamos alguna de las siguientes líneas de código:

```
automovil.encender; //notación de puntos  
automovil['encender']; //notación de corchetes
```

Veremos que no pasa absolutamente nada. Esto es porque la función no se está ejecutando, sino que se está accediendo a su contenido, es decir, a su código fuente. Para ver reflejado ésto, podemos imprimir lo anterior mediante la ejecución de la función **alert** mediante alguna de las siguientes líneas de código:

```
alert(automovil.encender);  
alert(automovil['encender']);
```

Lo anterior nos mostrará por pantalla un mensaje como el siguiente:

```
function (){  
  alert('automóvil encendido.');
```

Entonces, ¿Qué debemos hacer para ejecutar una función que está dentro de un objeto? Lo único que basta hacer es agregar como sufijo los paréntesis, ya que ésto le dice al navegador que ejecutaremos la función, lo correcto sería lo siguiente:

```
automovil.encender(); //notación de puntos  
automovil['encender'](); //notación de corchetes
```

Ejercicio guiado: Accediendo a las funciones de un objeto

Acceder a la función que se encuentra en el objeto con el nombre de “perro”, que se muestra en el siguiente bloque de código:

```
let perro = {  
  raza: 'Pastor Alemán',  
  origen: 'Alemania',  
  pelaje: 'Lanudo',  
  peso: '33kg',  
  edad: 12,  
  amigable: true,  
  sonidos: function(){  
    console.log('El perro ladra');  
  }  
};
```

- **Paso 1:** Se debe implementar cualquiera de las dos notaciones disponibles para acceder a una función dentro de un objeto, primero revisemos la función que se encuentra dentro del objeto, la cual es:

```
sonidos: function(){  
  console.log('El perro ladra');  
}
```

- **Paso 2:** Si se quiere acceder a esa llave del objeto que lleva por nombre “sonidos”, se puede implementar la notación mediante el punto más el nombre de la llave del objeto con los paréntesis para hacer el llamado y la ejecución de la función.

```
console.log(perro.sonidos());  
console.log(perro['sonidos']());
```

- **Paso 3:** Ahora implementamos la otra posibilidad de ejecutar a la función mediante el acceso al elemento o llave, implementando la notación mediante corchetes y agregando los paréntesis al final después de los corchetes, quedando:

```
console.log(perro.sonidos());  
console.log(perro['sonidos']());
```

- **Paso 4:** Ejecutar el código anterior obteniendo el mismo resultado para ambos métodos, siendo este:

```
El perro ladra
```

Modificación de propiedades o funciones dentro de un Objeto

La modificación de elementos dentro de un objeto se hace de manera similar a como lo haríamos con un array. Por ejemplo, si deseamos modificar el valor de la propiedad **patente** dentro del objeto **automovil**, podemos hacerlo de la siguiente forma:

```
automovil.patente = 'JJKX12';
```

Esta acción sobrescribe el valor de la propiedad sin importar su naturaleza, es decir, no importa si el tipo de dato no es igual al anterior o si intentamos modificar una función con una propiedad del tipo numérico (por ejemplo).

Ejercicio guiado: Modificando el objeto original

Realizar algunas modificaciones en el objeto original, como el peso, la edad y el elemento amigable mediante la notación punto, sobrescribiendo los valores originales por unos valores nuevos. Utilizamos el siguiente objeto:

```
let perro = {  
  raza: 'Pastor Alemán',  
  origen: 'Alemania',  
  pelaje: 'Lanudo',  
  peso: '33kg',  
  edad: 12,  
  amigable: true,  
  sonidos: function(){  
    console.log('El perro ladra');  
  }  
};
```

- **Paso 1:** Para modificar el peso, la edad y el elemento amigable con nuevos valores, se realizaría de la siguiente manera:

```
perro.peso = '30kg';  
perro.edad = 11;  
perro.amigable = false;  
  
console.log(perro);
```

- **Paso 2:** Utilizamos un console.log para visualizar el objeto nuevamente, obteniendo como resultado:

```
let perro = {  
  raza: 'Pastor Alemán',  
  origen: 'Alemania',  
  pelaje: 'Lanudo',  
  peso: '30kg',  
  edad: 11,  
  amigable: true,  
  sonidos: function(){  
    console.log('El perro ladra');  
  }  
};
```

Como se puede observar en el resultado, solo se modificaron los valores indicados mediante la notación del punto, sobrescribiendo esos valores en específicos por los nuevos. Por consiguiente, este método permitirá realizar la modificación directa de valores en los objetos sin intervenir en los otros elementos que posea la variable.

Espacio de nombres dentro de un Objeto

Las propiedades dentro de un objeto pueden contener a su vez objetos que pueden establecer cierta jerarquía definida por nosotros. Esta estructura, formalmente llamada espacio de nombres, puede ser accedida de manera muy sencilla en JavaScript. Por ejemplo, si quisiéramos agregar más detalles a la propiedad **marca** del objeto **automovil**, podríamos realizarlo de la siguiente manera:

```
var automovil = {  
  marca: {  
    nombre: 'Mazda',  
    origen: 'Japón'  
  },  
  modelo: '3 sport',  
  patente: 'LJKH63',  
  color: 'azul',  
  kilometraje: 15000,  
  usado: false,  
  encender: function(){  
    alert('automóvil encendido.');  }  
};
```

Ahora, para acceder al nombre de la marca del automóvil, se podría realizar de alguna de las siguientes formas:

```
automovil.marca.nombre;    //notación de puntos  
automovil['marca']['nombre'] //notación de corchetes
```

Ejercicio guiado: Agregando objetos a un objeto

Utilizar el siguiente objeto perro, al cual hemos agregado algunas propiedades como propietario y lugar.

```
let perro = {
  propietario: {
    nombre: 'Juan',
    edad: 34,
    lugar: {
      pais: 'Chile',
      ciudad: 'Santiago de Chile'
    },
  },
  raza: 'Pastor Alemán',
  origen: 'Alemania',
  pelaje: 'Lanudo',
  peso: '30kg',
  edad: 11,
  amigable: true,
  sonidos: function(){
    console.log('El perro ladra');
  }
};
```

Como podemos observar en el objeto mostrado, se modificó la variable original agregando otros objetos. Ahora, intentemos acceder y mostrar los valores pertenecientes al propietario, como el nombre y el país donde reside.

Si lograste hacerlo, muy bien, buen trabajo. Si no lo lograste, no te preocupes, igualmente vamos a realizar el ejercicio paso a paso:

- **Paso 1:** Para poder acceder a los valores indicados, se utilizará la notación mediante punto y se mostrará el resultado con un `console.log`, por lo tanto:

```
console.log(perro.propietario.nombre);
console.log(perro.propietario.lugar.pais);
```

- **Paso 2:** Ahora implementamos la notación mediante el uso de corchetes, mostrando el resultado a través de un `console.log`, eso se logra de la siguiente manera:

```
console.log(perro['propietario']['nombre']);
console.log(perro['propietario']['lugar']['pais']);
```

- **Paso 3:** Ejecutamos los códigos anteriores, obteniendo el mismo resultado para ambos casos:

Juan
Chile

Ejercicio propuesto (6)

Para el objeto dado, realice las siguientes operaciones implementando la notación de punto para poder acceder y mostrar los elementos: nombre, edad y peso del alumno. Además el nombre y la ciudad del representante. Luego ejecuta y muestra la función que se encuentra en el elemento denominado rendimiento. Utiliza el `console.log` para obtener los resultados en el navegador web.

```
let alumno = {
  representante: {
    nombre: 'Maria',
    edad: 49,
    lugar: {
      pais: 'Chile',
      ciudad: 'Santiago de Chile'
    },
  },
  nombre: 'Manuel',
  apellido: 'Perez',
  peso: '45kg',
  edad: 10,
  amigable: true,
  rendimiento: function(){
    console.log('Muy buen alumno');
  }
};
```

Uso del operador *this* dentro de un Objeto

Si deseamos que el código escrito realice lo que tenemos en mente y respete nuestras intenciones, es imprescindible que siempre nos aseguremos que cada uno de los elementos acceda correctamente a los componentes. Un error muy común es escribir variables con el mismo nombre una y otra vez dentro de contextos distintos, lo que induce a confusiones. Por esto, existe un operador llamado **this**. Esta palabra reservada, nos permite acceder a elementos que existen sólo dentro del contexto en el cual estamos trabajando. Por ejemplo:


```
this.elemento.elemento;
```

Para comprender este concepto, modificaremos el objeto **automovil** que creamos y utilizamos anteriormente con los siguientes valores:

```
let automovil = {  
  marca: {  
    nombre: 'Mazda',  
    origen: 'Japón'  
  },  
  modelo: '3 sport',  
  patente: 'LJKH63',  
  color: 'azul',  
  kilometraje: 15000,  
  usado: false,  
  encender: function(){  
    alert('automóvil ' + this.marca.nombre + ' ' + this.modelo + '  
encendido.');  }  
};
```

Ahora, si ejecutamos el método **encender** del objeto **automovil**:

```
automovil.encender();
```

Obtendremos como salida el siguiente mensaje:

```
automóvil Mazda 3 sport encendido.
```

Como podemos ver dentro de la función **encender**, utilizamos el operador **this** para acceder al contexto del objeto **automovil** y así poder utilizar las propiedades **marca** y **modelo**. Para este caso, de no haber utilizado **this**, obtendremos un error de ejecución, ya que dichas propiedades o variables no existen.

Si, por el contrario, se realiza la declaración dentro de la función **encender** sin el uso de **this** y se cumple la condición de que ya existan variables declaradas con los mismos nombres, como por ejemplo **marca** y **modelo**, como muestra el siguiente código:

```
var marca = {nombre: 'Honda'};
var modelo = 'Civic';
var automovil = {
  marca: {
    nombre: 'Mazda',
    origen: 'Japón'
  },
  modelo: '3 sport',
  patente: 'LJKH63',
  color: 'azul',
  kilometraje: 15000,
  usado: false,
  encender: function(){
    alert('automóvil ' + marca.nombre + ' ' + modelo + ' ' +
encendido.');
```

El resultado de la ejecución sería distinto, mostrándonos lo siguiente:

```
automóvil Honda Civic encendido.
```

Si bien no profundizaremos en la programación orientada a objetos en esta unidad, es importante abordar un concepto muy ligado a los objetos: las clases. La principal ventaja de las clases en JavaScript, es proveer una sintaxis más clara y simple para crear objetos. Para comenzar a internalizar estos conceptos, puedes consultar el documento **Material Apoyo Lectura - Objetos como instancia de una clase**, ubicado en “Material Complementario”.

Como vimos, la utilización del operador `this` siempre será importante para poder acceder a elementos precisos dentro del mismo contexto, por tanto, realizaremos otro ejercicio para practicar su uso.

Ejercicio guiado: Utilizando el operador this

Crear una función dentro del objeto que muestre los valores de raza, edad y el nombre del propietario. La función debe ser parte de un elemento denominado “datos” y se debe mostrar el siguiente mensaje: “La raza del perro es pastor Alemán, tiene una edad de 11 años y el propietario es Juan”. El objeto a trabajar es:

```
let perro = {  
  propietario: {  
    nombre: 'Juan',  
    edad: 34,  
    lugar: {  
      pais: 'Chile',  
      ciudad: 'Santiago de Chile'  
    },  
  },  
  raza: 'Pastor Alemán',  
  origen: 'Alemania',  
  pelaje: 'Lanudo',  
  peso: '30kg',  
  edad: 11,  
  amigable: true,  
};
```

- **Paso 1:** Crear una carpeta en tu lugar de trabajo favorito y dentro de ella crea dos archivos, un index.html y un script.js.
- **Paso 2:** En el archivo script.js, utilizar el objeto facilitado en el enunciado y realizar las modificaciones correspondientes para agregar la función dentro del objeto. En este caso, el nombre del elemento que contendrá la función será “datos”, donde pasaremos una función y dentro de ella el mensaje en un console.log para mostrar en el navegador web el resultado:

```
datos: function(){ console.log(``) } };
```

- **Paso 3:** Agregar el elemento creado dentro del objeto con el mensaje en el console.log, implementando el operador this para hacer la referencia a los elementos del objeto que deseamos mostrar dentro de esa función:

```
let perro = {
  propietario: {
    nombre: 'Juan',
    edad: 34,
    lugar: {
      pais: 'Chile',
      ciudad: 'Santiago de Chile'
    },
  },
  raza: 'Pastor Alemán',
  origen: 'Alemania',
  pelaje: 'Lanudo',
  peso: '30kg',
  edad: 11,
  amigable: true,
  datos: function(){
    console.log(`La raza del perro es ${this.raza}, tiene una edad de
    ${this.edad} años y el propietario es ${this.propietario.nombre}`);
  }
};
```

- **Paso 4:** Utilizar la notación de punto, llamemos al elemento que contiene la función para poder ejecutarla y se muestra el resultado:

```
perro.datos();
```

- **Paso 5:** Ejecutar el código anterior en el navegador web y el resultado debería ser:

```
La raza del perro es Pastor Alemán, tiene una edad de 11 años y el
propietario es Juan.
```

Ejercicio propuesto (7)

Crear una función que muestre los valores del representante, como nombre y país. Además, el nombre y apellido del alumno en un mensaje que indique: "El representante del alumno Manuel Pérez es Maria, quien reside en Chile". La función debe estar en un elemento denominado mensaje y utilizando el operador this, trae los valores necesarios a ser mostrados en el mensaje final dentro de la función.

```
let alumno = {
  representante: {
    nombre: 'Maria',
    edad: 49,
    lugar: {
      pais: 'Chile',
      ciudad: 'Santiago de Chile'
    },
  },
  nombre: 'Manuel',
  apellido: 'Perez',
  peso: '45kg',
  edad: 10,
  amigable: true,
};
```

Ejercicio guiado: Integrando HTML

Para profundizar en las bondades de utilizar e iterar sobre colecciones de datos. Se solicita recorrer un objeto y mostrar en una tabla un documento HTML. Lo que debemos mostrar, será la raza, origen, pelaje, peso y edad del arreglo perros:

```
let perros = [{
  raza: 'Pastor Alemán',
  origen: 'Alemania',
  pelaje: 'Lanudo',
  peso: '33kg',
  edad: 12
},
{
  raza: 'Poodle',
  origen: 'Francia',
  pelaje: 'Lanudo',
  peso: '20kg',
  edad: 14
},
];
```

- **Paso 1:** Crear una carpeta en tu lugar de trabajo favorito y dentro de ella crea dos archivos, un index.html y un script.js. En el archivo index.html utilizaremos la

estructura básica del documento más una tabla, que nos permitirá agregar dentro la información del objeto:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<meta http-equiv="X-UA-Compatible" content="ie=edge">
<title>Integrando HTML</title>
<style>
table, th, td {
  border: 1px solid black;
  border-collapse: collapse;
}
</style>
</head>
<body>
<table id="cuerpo-tabla">
</table>

<script src="script.js"></script>
</body>
</html>
</html>
```

- **Paso 2:** Dentro del archivo script.js, declarar el objeto y una variable con las cabeceras de la tabla:

```
let perros = [{
  raza: 'Pastor Alemán',
  origen: 'Alemania',
  pelaje: 'Lanudo',
  peso: '33kg',
  edad: 12
},
{
  raza: 'Poodle',
  origen: 'Francia',
  pelaje: 'Lanudo',
  peso: '20kg',
  edad: 14
},
]
```

```
];
```

```
var texto =  
"<tr><th>Raza</th><th>Origen</th><th>Pelaje</th><th>Peso</th><th>Edad</th><th></tr>";
```

- **Paso 3:** Dentro del archivo script.js, declarar el objeto y generar la lógica para recorrer los elementos raza, origen, pelaje, peso y edad:

```
var texto =  
"<tr><th>Raza</th><th>Origen</th><th>Pelaje</th><th>Peso</th><th>Edad</th><th></tr>";
```

```
for (var i = 0; i < perros.length; i++) {  
}
```

- **Paso 4:** Generar una estructura de tablas que almacene la fila del elemento en la posición [i] en la variable `texto`:

```
var texto =  
"<tr><th>Raza</th><th>Origen</th><th>Pelaje</th><th>Peso</th><th>Edad</th><th></tr>";
```

```
for (var i = 0; i < perros.length; i++) {  
    texto += `<tr>  
        <td>${perros[i].raza}<td>  
        <td>${perros[i].origen}<td>  
        <td>${perros[i].pelaje}<td>  
        <td>${perros[i].peso}<td>  
        <td>${perros[i].edad}<td>  
    </tr>`;  
}
```

- **Paso 5:** Imprimir la información en el HTML, agregando un id `cuerpo-tabla` donde se cargará el texto:

```
var texto =  
"<tr><th>Raza</th><th>Origen</th><th>Pelaje</th><th>Peso</th><th>Edad</th><th></tr>";
```

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

```
texto += `<tr>
    <td>${perros[i].raza}<td>
    <td>${perros[i].origen}<td>
    <td>${perros[i].pelaje}<td>
    <td>${perros[i].peso}<td>
    <td>${perros[i].edad}<td>
</tr>`;
}
document.getElementById("cuerpo-tabla").innerHTML = texto;
```

Esto da como resultado lo siguiente en el navegador web:

Raza	Origen	Pelaje	Peso	Edad
Pastor Alemán	Alemania	Lanudo	33kg	12
Poodle	Francia	Lanudo	20kg	14

Imagen 5. Tabla del objeto perros
Fuente: Desafío Latam

Ejercicio propuesto (8)

A partir del objeto dado a continuación, crear un objeto con cada uno de los elementos. Finalmente, mostrar una tabla en HTML con sus propiedades:

Nombre	Apellido	Peso	Altura	Edad
Pedro	Soto	66kg	1.80m	25
Ana	Gutierrez	57kg	1.60m	27
Mario	Torres	80kg	1.82m	33

Tabla 2. Objeto alumnos
Fuente: Desafío Latam

A continuación veremos algunos métodos útiles para recorrer objetos. Antes de ir de lleno al código, puedes consultar el documento **Material Apoyo Lectura - Iterando un objeto**, ubicado en “Material Complementario”, donde se explica a través de un ejercicio cómo se originan estos métodos y las propiedades del objeto en las que se basan las iteraciones.

Iterando sobre objetos

Competencia

- Realizar operaciones iterativas sobre elementos de un objeto utilizando estos recursos para resolver un problema.

Introducción

Como hemos mencionado anteriormente, una de las principales ventajas de la programación es la posibilidad de simplificar y dar legibilidad al código mediante bucles y repeticiones. Conocer la sintaxis para realizar estas tareas y el alcance de cada uno de los elementos que nos permiten iterar es clave para aprovechar los recursos del lenguaje.

Así como hemos abordado algunos métodos para recorrer arreglos, los objetos también nos proveen herramientas propias para acceder a la información que contienen, de forma sencilla. A continuación, veremos en detalle estos conceptos.

Métodos para recorrer un Objeto

Como hemos visto, los objetos son estructuras de datos muy útiles en JavaScript, internamente el lenguaje administra todo mediante objetos. Para recorrer sus elementos, podemos recurrir a sus propios métodos, los cuales veremos a continuación:

Object.keys

Crea un array con las propiedades del objeto, es decir, el método “keys” devuelve un objeto del arreglo con las claves de ese arreglo. Por ejemplo, si tenemos un objeto con los elementos “manzanas, naranjas y peras” y cada uno de ellos con valores asignados, podemos implementar el `Object.keys` del objeto para retornar un arreglo con los elementos o propiedades de esos objeto, más no los valores que tiene cada propiedad:

```
var compras = {  
  manzana: 2,  
  naranjas: 5,  
  peras: 10,  
};  
  
var keys = Object.keys(compras);  
console.log(keys);
```

Al ejecutar ese código, el resultado sería:

```
(3) [ "manzana", "naranjas", "peras" ]
```

Object.values

Crea un array con los valores del objeto. En otras palabras, el método “values” retorna un arreglo con los valores correspondientes a las propiedades dispuestas en un objeto. Las propiedades retornan en el mismo orden a como lo haría un ciclo repetitivo `for...in`. Por lo tanto, este objeto construirá un nuevo arreglo pero con los valores de la propiedades. Si continuamos con el objeto utilizado anteriormente denominado `compras`, y aplicamos el `Object.values` a ese objeto:

```
var compras = {  
  manzana: 2,  
  naranjas: 5,  
  peras: 10,  
};  
  
var values = Object.values(compras);  
console.log(values);
```

Al ejecutar el código anterior y mostrar el resultado en la consola del navegador web mediante un `console.log`, el resultado es:

```
(3) [ 2, 5, 10 ]
```

Object.entries

Devuelve una matriz de pares, en donde el primer elemento del arreglo es la propiedad del objeto y el otro el valor, es decir, creará un arreglo por cada elemento-valor del objeto. Para detallar mejor el funcionamiento de este método, realicemos un ejercicio con el objeto que trabajamos en los ejemplos pasados, agregando el método al objeto, guardando en una variable el resultado y finalmente mostrando esa variable mediante un `console.log`:

```
var compras = {  
  manzana: 2,  
  naranjas: 5,  
  peras: 10,  
};  
  
var entries = Object.entries(compras);  
console.log(entries);
```

Al ejecutar el código anterior, se obtendría como resultado:

```
[[ "manzana", 2 ],[ "naranjas", 5 ],[ "peras", 10 ]]
```

Ejercicio guiado: Métodos para recorrer un objeto

A continuación vamos a experimentar los métodos vistos para obtener las claves, valores y matriz de pares del objeto perro con el que hemos estado trabajando:

```
let perro = {  
  propietario: {  
    nombre: 'Juan',  
    edad: 34,  
    lugar: {  
      país: 'Chile',  
      ciudad: 'Santiago de Chile'  
    },  
  },  
  raza: 'Pastor Alemán',  
  origen: 'Alemania',  
  pelaje: 'Lanudo',  
  peso: '30kg',  
  edad: 11,  
  amigable: true,  
};
```

- **Paso 1:** En nuestra consola, creamos el arreglo y para obtener las claves, utilizamos el método `keys()` sobre el objeto 'perro', de la siguiente manera:

```
var keys = Object.keys(perro);  
console.log(keys);
```

Esto retorna un arreglo con las claves que posee el objeto:

```
(7) ["propietario", "raza", "origen", "pelaje", "peso", "edad",  
"amigable"]
```

- **Paso 2:** Para obtener los valores, utilizamos el método `values()` sobre el objeto 'perro', de la siguiente manera:

```
var values = Object.values(perro);  
console.log(values);
```

Esto retorna un arreglo con los valores que posee el objeto:

```
(7) [{...}, "Pastor Alemán", "Alemania", "Lanudo", "30kg", 11, true]
0: {nombre: "Juan", edad: 34, lugar: {...}}
1: "Pastor Alemán"
2: "Alemania"
3: "Lanudo"
4: "30kg"
5: 11
6: true
```

- **Paso 3:** Finalmente, para obtener la matriz de pares: clave, valor, utilizamos el método `entries()`:

```
var entries = Object.entries(perro);
console.log(entries);
```

Que retorna la siguiente información:

```
(7) [Array(2), Array(2), Array(2), Array(2), Array(2), Array(2),
Array(2)]
0: (2) ["propietario", {...}]
1: (2) ["raza", "Pastor Alemán"]
2: (2) ["origen", "Alemania"]
3: (2) ["pelaje", "Lanudo"]
4: (2) ["peso", "30kg"]
5: (2) ["edad", 11]
6: (2) ["amigable", true]
```

Ejercicio propuesto (9)

Para el objeto que se mostrará a continuación, obtener un arreglo con las propiedades, otro arreglo con los valores de las propiedades y finalmente un arreglo que contenga los arreglos de las propiedades y valores del objeto.

```
var mascotas = {perros: "Pastor Aleman", gatos: 5, aves: "guacamayas"};
```

Sets

En ES6 se introdujo el objeto de la clase **Set**, que es una colección de elementos únicos en un listado, veamos un par de ejemplos de cómo inicializar un objeto de la clase set

Existen diferentes maneras de declarar un set:

- Declaración de un set vacío:

```
// set vacio
var set1 = new Set();
console.log(set1);
```

Resultado:

```
Set []
```

- Declaración pasando un string con caracteres repetidos:

```
var set2 = new Set("hoola");
console.log(set2);
```

El resultado:

```
Set(4) [ "h", "o", "l", "a" ]
```

Si te das cuenta en el **set2** al pasarle valores repetidos el **Set** los eliminó automáticamente.

- Otra manera es pasarle un array al set.

```
var set3 = new Set([1,2,3,4,5]);
console.log(set3);
```

Siendo el siguiente el resultado:

```
Set(5) [ 1, 2, 3, 4, 5 ]
```

Los sets al ser objetos heredan diferentes métodos por defecto. Veamos algunos que pueden ser útiles en nuestro día a día como programadores.

Método add

Agrega un elemento a la lista.

```
var set1 = new Set();

set1.add(1);
set1.add(2);
set1.add(3);
console.log(set1);
```

El resultado es el siguiente:

```
Set(3) [ 1, 2, 3 ]
```

Método delete

Elimina un elemento a la lista:

```
var set1 = new Set([1,2,3,4,5]);
set1.delete(5)
console.log(set1);
```

Este será el resultado:

```
Set(4) [ 1, 2, 3, 4 ]
```

Método forEach

Recorre la lista:

```
var set1 = new Set([1,2,3,4,5]);
set1.forEach(function(item) {
    console.log(item);
});
```

Siendo este el resultado:

```
1  
2  
3  
4  
5
```

Ejercicio guiado: Crear un arreglo desde un mensaje

Se solicita crear un nuevo arreglo partiendo del siguiente mensaje: "Arreglo de string". Luego, eliminar los seis (6) últimos elementos del arreglo formado. Posteriormente, agregar al final del arreglo "simple". Finalmente, mostrar todo el contenido del arreglo implementando el objeto Set y sus métodos. Para poder dar respuesta a la solicitud, sigamos los siguientes pasos:

- **Paso 1:** Crear una carpeta en tu lugar de trabajo favorito y dentro de ella crear dos archivos, un index.html y un script.js.
- **Paso 2:** En el archivo script.js, inicializar el arreglo utilizando el objeto set, lo cual sería:

```
let arreglo = new Set("Arreglo de string");  
console.log(arreglo);
```

- **Paso 3:** Al ejecutar el código anterior y ver el resultado en la consola del navegador web, el resultado sería:

```
0: "A"  
1: "r"  
2: "e"  
3: "g"  
4: "l"  
5: "o"  
6: " "  
7: "d"  
8: "s"  
9: "t"  
10: "i"  
11: "n"
```


- **Paso 4:** En el resultado anterior, se puede observar como el objeto set elimina todos los caracteres que se encuentren repetidos. Ahora, se procede a eliminar los últimos seis (6) elementos, siendo estos: " ,d,s,t,i,n". Aquí se debe implementar el método delete del objeto set.

```
arreglo.delete(" ");  
arreglo.delete("d");  
arreglo.delete("s");  
arreglo.delete("t");  
arreglo.delete("i");  
arreglo.delete("n");  
console.log(arreglo);
```

- **Paso 5:** Ejecutar en el navegador web la instrucción anterior, el resultado quedaría:

```
Set(6) [ "A", "r", "e", "g", "l", "o" ]
```

- **Paso 6:** Implementar el método add y agregar la palabra "simple" al arreglo resultante después de eliminar los elementos indicados. Quedando el código de la siguiente manera:

```
arreglo.add("s");  
arreglo.add("i");  
arreglo.add("m");  
arreglo.add("p");  
arreglo.add("l");  
arreglo.add("e");  
console.log(arreglo);
```

- **Paso 7:** Obteniendo como resultado al ejecutar el código anterior:

```
Set(10) [ "A", "r", "e", "g", "l", "o", "s", "i", "m", "p" ]
```

- **Paso 8:** Mostrar todo el arreglo con el método forEach:

```
arreglo.forEach(function(elementos) {  
    console.log(elementos);  
});
```

- **Paso 9:** Al ejecutar el código anterior, el resultado sería:

A
r
g
l
o
s
i
m
p

Ejercicio propuesto (10)

Crear un nuevo arreglo partiendo del siguiente mensaje: "06-07-1985" y eliminar los cuatro (4) últimos elementos del arreglo formado. Posteriormente, agregar al final del arreglo "julio". Finalmente, mostrar todo el contenido del arreglo implementando el objeto Set y sus métodos.

¿Objetos o Arreglos?

Como vimos durante la lectura podemos acceder a los valores de un objeto usando notación de puntos o corchetes, lo cual puede hacernos pensar que un objeto es un array, pero por muy parecido que se vean, son diferentes. Un objeto se utiliza para representar una "cosa" por ejemplo un auto, mesa, persona, animal, etc. y su definición es {llave: valor}. En cambio un array se utiliza para crear un listado de elementos que no es necesario definir llave o valor, aunque sí posee un índice de las posiciones de estos elementos, otra diferencia son los métodos que posee el lenguaje para trabajar con cada una de las estructuras.

Como pudiste apreciar la forma de operar un array y un objeto son muy diferentes, a continuación podrás ver una tabla resumen para ver sus diferencias:

Tabla resumen

Diferencia	Objeto	Array
Uso	Sirven para definir una "cosa" ej: autos, animales, personas, etc..	Sirven para listar elementos
Declaración vacía	<code>var obj= {}, var obj = new Obj()</code>	<code>var array = []</code>
Declaración con elementos	<code>var obj= {a:1,b:2}, var obj = new Obj(1,2)</code>	<code>var array = ["1", "2", 3, 4]</code>
Acceso a valores	<code>obj.a, obj["a"]</code>	<code>array[0]</code> donde 0 es la posición del elemento
Agregar valor	<code>obj.c = "nuevo valor"</code>	<code>array.push("nuevo valor")</code> o <code>array.unshift("nuevo valor")</code>
Eliminar valor	<code>delete obj.c</code>	<code>array.pop()</code> o <code>array.shift()</code>

Tabla 3. Diferencias entre Arrays y Objetos

Fuente: Desafío Latam

Para profundizar en tus conocimientos sobre cómo trabajar con arreglos y objetos, puedes consultar el documento **Material Apoyo Lectura - Métodos para trabajar con arreglos y objetos**, ubicado en "Material Complementario". En este documento podrás consultar algunos de los métodos esenciales para agregar y quitar elementos de estas estructuras de datos.

Resumen

En esta lectura hemos visto cómo trabajar con estructuras de datos sencillas, como son los arreglos en JavaScript, los cuales nos permiten acceder a grandes cantidades de datos de forma simple y además, son de gran utilidad cuando necesitamos resolver problemas que requieran trabajar con colecciones de elementos del mismo tipo.

Por otro lado, revisamos los objetos, que son una estructura de datos un poco más compleja que los arreglos y nos permite trabajar con métodos propios, para agregar, eliminar, modificar y recorrer elementos, facilitando el desarrollo y su legibilidad.

Solución de los ejercicios propuestos

1. Crea una variable del tipo array que contenga los siguientes datos: "Perro", "Gato", ave: "guacamaya", 2020, "erizo", lugar: "Chile", ciudad: "Santiago de Chile". Luego muestra el resultado en la consola del navegador.

```
let datosPersonales =  
[ 'Perro', 'Gato', {ave: 'guacamaya'}, 2020, 'erizo', {lugar: 'Chile'}, {ciudad: '  
Santiago de Chile' }];
```

2. Escribe las coordenadas del array que permitan mostrar los valores 1, 13 y 34.

```
matriz3x3[0][0];  
matriz3x3[1][2];  
matriz3x3[2][1];
```

3. Para el arreglo que se presenta a continuación, muestra por la consola del navegador web la cantidad de elementos totales que contiene el arreglo, luego reduce en un elemento el arreglo actual, es decir, que pase de 5 elementos a 4, por último, añade un nuevo elemento en la posición 7 del arreglo denominado: "Express".

```
let datosWeb = ["JavaScript", "VueJS", "AngularJS", "ReactJS", "NodeJS"];  
console.log(datosWeb.length);  
datosWeb.length = 4;  
datosWeb[7] = "Express"
```

4. En base al siguiente código, escribe un ciclo que acceda y muestre con un console.log cada uno de los elementos dispuestos en el siguiente arreglo mediante el ciclo repetitivo for:

```
for (var i = 0; i < mascotas.length; i++) {  
  console.log(mascotas[i]);  
};
```

5. Accede y muestra mediante un console.log, cada uno de los elementos dispuestos en el siguiente arreglo mediante los ciclos repetitivos *for...in* y *for...of*:

```
for (let animales in mascotas){  
  console.log(animales);  
}
```

```
};  
for (let animal of mascotas){  
    console.log(animal);  
};
```

6. Para el objeto dado, realice las siguientes operaciones implementando la notación de punto para poder acceder y mostrar los elementos: nombre, edad y peso del alumno. Además el nombre y la ciudad del representante. Luego ejecuta y muestra la función que se encuentra en el elemento denominado rendimiento. Utiliza el console.log para obtener los resultados en el navegador web.

```
let alumno = {  
  representante: {  
    nombre: 'Maria',  
    edad: 49,  
    lugar: {  
      pais: 'Chile',  
      ciudad: 'Santiago de Chile'  
    },  
  },  
  nombre: 'Manuel',  
  apellido: 'Perez',  
  peso: '45kg',  
  edad: 10,  
  amigable: true,  
  rendimiento: function(){  
    console.log('Muy buen alumno');  
  }  
};  
  
console.log(alumno.nombre);  
console.log(alumno.edad);  
console.log(alumno.peso);  
console.log(alumno.representante.nombre);  
console.log(alumno.representante.lugar.ciudad);  
console.log(alumno.rendimiento());
```

7. A partir del objeto dado, crear una función que muestre los valores del representante, como nombre y país, además, el nombre y apellido del alumno en un mensaje que indique: "El representante del alumno Manuel Pérez es Maria, quien reside en Chile". La función debe estar en un elemento denominado mensaje y utilizando el operador `this`, trae los valores necesarios a ser mostrados en el mensaje final dentro de la función.

```
let alumno = {
  representante: {
    nombre: 'Maria',
    edad: 49,
    lugar: {
      pais: 'Chile',
      ciudad: 'Santiago de Chile'
    },
  },
  nombre: 'Manuel',
  apellido: 'Perez',
  peso: '45kg',
  edad: 10,
  amigable: true,
  mensaje: function(){
    return `El representante del alumno ${this.nombre} ${this.apellido}
es ${this.representante.nombre}, quien reside en
${this.representante.lugar.pais}`
  }
};

console.log(alumno.mensaje());
```

8. A partir del objeto dado a continuación, crear un objeto con cada uno de los elementos. Finalmente, mostrar una tabla en HTML con sus propiedades:

HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<meta http-equiv="X-UA-Compatible" content="ie=edge">
<title>Ejercicio 7</title>
```

```
<style>
table, th, td {
  border: 1px solid black;
  border-collapse: collapse;
}
</style>
</head>
<body>
<table id="cuerpo-tabla">
</table>

<script src="script.js"></script>
</body>
</html>
</html>
```

JS

```
let alumnos = [{
  nombre: 'Pedro',
  apellido: 'Soto',
  peso: '66kg',
  altura: '1.80m',
  edad: 25
},
{
  nombre: 'Ana',
  apellido: 'Gutierrez',
  peso: '57kg',
  altura: '1.60m',
  edad: 27
},
{
  nombre: 'Mario',
  apellido: 'Torres',
  peso: '80kg',
  altura: '1.82m',
  edad: 33
},
];

var texto =
"<tr><th>Nombre</th><th>Apellido</th><th>Peso</th><th>Altura</th><th>Eda
```

```
d</th></tr>";

    for (var i = 0; i < alumnos.length; i++) {
        texto += `<tr>
            <td>${alumnos[i].nombre}<td>
            <td>${alumnos[i].apellido}<td>
            <td>${alumnos[i].peso}<td>
            <td>${alumnos[i].altura}<td>
            <td>${alumnos[i].edad}<td>
        </tr>`;
    }
    document.getElementById("cuerpo-tabla").innerHTML = texto;
```

9. Para el objeto que se mostrará a continuación, obtener un arreglo con las propiedades, otro arreglo con los valores de las propiedades y finalmente un arreglo que contenga los arreglos de las propiedades y valores del objeto.

```
var mascotas = {
    perros: "Pastor Aleman",
    gatos: 5,
    aves: "guacamayas"
};
var keys = Object.keys(mascotas);
console.log(keys);
var values = Object.values(mascotas);
console.log(values);
var entries = Object.entries(mascotas);
console.log(entries);
```

10. Crea un nuevo arreglo partiendo del siguiente mensaje: "06-07-1985". Luego, eliminar los cuatro (4) últimos elementos del arreglo formado. Posteriormente, agregar al final del arreglo "julio". Finalmente mostrar todo el contenido del arreglo implementando el objeto Set y sus métodos.


```
var fecha = new Set("06-07-1985");
console.log(fecha);
fecha.delete("1");
fecha.delete("9");
fecha.delete("8");
fecha.delete("5");
console.log(fecha);
fecha.add("j");
fecha.add("u");
fecha.add("l");
fecha.add("i");
fecha.add("o");
console.log(fecha);
fecha.forEach(function(element) {
    console.log(element);
});
```