Arrays y objetos (Parte II)

Métodos transformadores para añadir o eliminar elementos

Competencias

- Reconocer los métodos integrados que agregan o eliminan elementos de un objeto o arreglo, para utilizar adecuadamente las características del lenguaje JavaScript.
- Desarrollar algoritmos utilizando agregación y/o eliminación de elementos de un arreglo acorde al lenguaje JavaScript para resolver un problema.

Introducción

Como ya vimos en capítulos anteriores, los arreglos y objetos nos permiten gestionar datos o elementos de nuestros desarrollos con JavaScript, entregándonos algunas herramientas para poder manipular estos elementos de manera eficiente y para una cantidad importante de operaciones que son una constante en el desarrollo de aplicaciones. Es por esto, que haremos un recorrido por algunas de estas funciones o métodos, los cuales nos harán más fácil la tarea de codificar.

Para revisar el resultado de cada ejecución de los métodos que se verán a continuación, utilizaremos la consola JavaScript que se abre en cada navegador con las herramientas de desarrollador (en Google Chrome ésta se abre a través de la tecla F12, en la pestaña Console).

A continuación, veremos aquellos métodos que nos permiten manipular objetos y/o arreglos, es decir, agregar y eliminar elementos. Esto tiene gran utilidad en el trabajo diario de un desarrollador, ya que son herramientas fundamentales para resolver algoritmos medianamente complejos.

Métodos Integrados con Arreglos

Como ya mencionamos anteriormente, los arreglos son objetos en JavaScript. Esto quiere decir que cada arreglo que creamos hereda por defecto, propiedades y métodos:

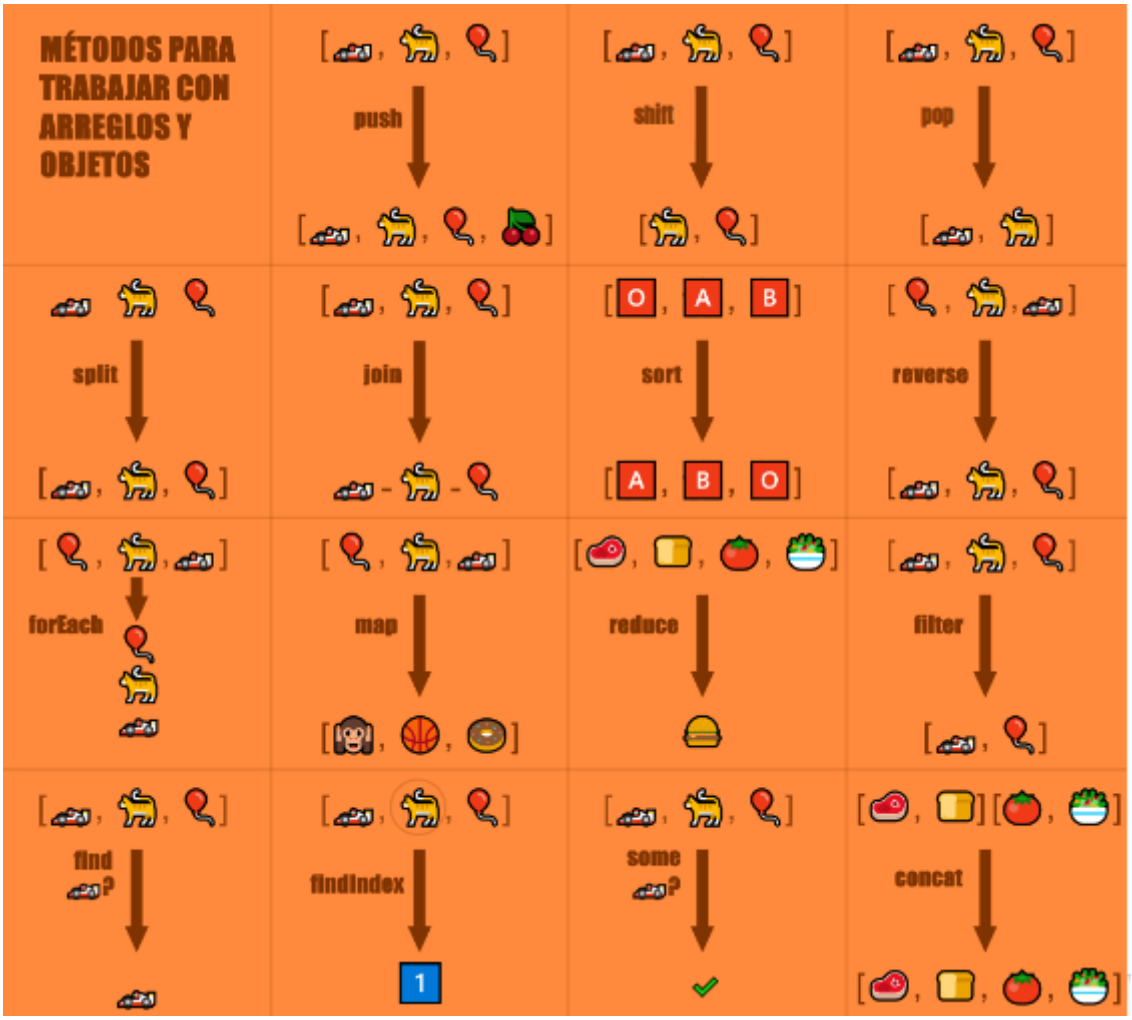

Imagen 1. Diagrama de algunos métodos para trabajar con arreglos y objetos en JavaScript.
Fuente: Desafío Latam

Los métodos que muestra la imagen 1 nos pueden ayudar a cumplir ciertas tareas, que por otras vías, serían mucho más difíciles de alcanzar. Para ver ésto, podemos ejecutar la siguiente línea de código y observar su resultado en la consola del navegador:

```
var arreglo = []
console.log(arreglo);
```

Dicho resultado será como el siguiente:

```
[]
length: 0
__proto__: Array(0)
```

Además de la propiedad length que ya vimos en el capítulo de arrays, podemos ver que hay otra propiedad llamada **proto**. Esta propiedad privada corresponde a características heredadas de **Object**, el cual es el objeto padre de todo array. Si desplegamos la lista de elementos que contiene **proto**, podemos ver que en su interior hay una cantidad importante de objetos y métodos como **forEach, join, filter, map, pop**, entre otros.

Cada uno de estos métodos están presentes y se heredan para ayudarnos a gestionar los arrays de una u otra manera, ya sea para eliminar alguno de sus elementos, ordenarlos, filtrarlos, etc. A continuación, veremos algunos de ellos y cómo podemos utilizarlos.

Método push

Para realizar inserciones de objetos dentro de un array, podemos valernos del método push, el cual nos permite agregar uno o más elementos al final de nuestro array, como muestra la siguiente imagen:

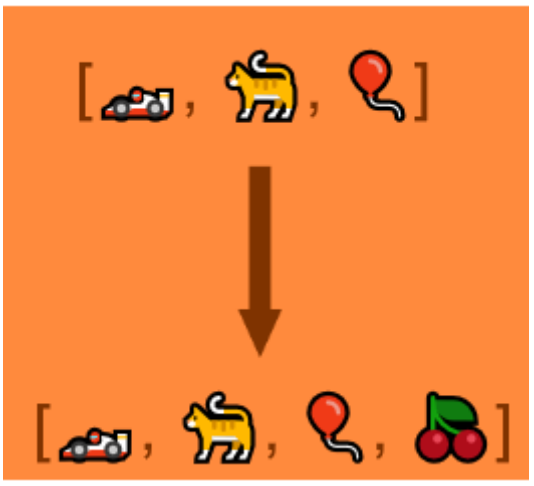

Imagen 2. Diagrama de ejemplificación del método push().
Fuente: Desafío Latam

Veamos un ejemplo para entender mejor este método. Agregar tres nuevos elementos al arrays original llamado `amigos` utilizando el método push. Estos elementos serán: "Gary, Arturo y Alexis", el arreglo original denominado amigos está conformado por: "Erick, Cristian, Max y Claudia". Entonces, para agregar al final del arreglo original los nuevos elementos, implementamos el método push.

```
var amigos = ["Erick", "Cristian", "Max", "Claudia"];
amigos.push("Gary", "Arturo", "Alexis");
console.log(amigos);
```

Al ejecutar el código anterior, se generará una salida como la siguiente:

```
(7) ["Erick", "Cristian", "Max", "Claudia", "Gary", "Arturo", "Alexis"]
```

Realicemos ahora otro ejercicio, pero esta vez trabajando con objetos dentro de un arreglo. Esto es algo muy común y que se encuentra a cada momento cuando se trabaja con interfaces de información.

Ejercicio guiado: Método push

Incorporar dos objetos usando el método push. En este caso, el arreglo principal contiene los objetos: {auto: "Peugeot", color: "rojo" },{auto: "Mazda", color: "azul" },{auto: "BMW", color: "negro" }.

- ***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 para poder pasar objetos a un arreglo, simplemente se debe colocar el objeto completo dentro de los paréntesis del push y separar por coma si se va a agregar otro objeto.

```
var autos = [{auto: "Peugeot", color: "rojo" },{auto: "Mazda", color:
"azul" },{auto: "BMW", color: "negro" }];
autos.push({auto: "Suzuki", color: "verde" },{auto: "Chevrolet", color:
"amarillo" });
console.log(autos);
```

- ***Paso 3*:** Ejecutar el código anterior en el navegador, se generará lo siguiente:

```
(5) [{auto: "Peugeot", color: "rojo" },{auto: "Mazda", color: "azul"
},{auto: "BMW", color: "negro" },{auto: "Suzuki", color: "verde"
},{auto: "Chevrolet", color: "amarillo" }]
```

Para mayores detalles y especificaciones del **método push**, puede visitar el siguiente enlace

Ejercicio propuesto (1)

Partiendo del arreglo mostrado a continuación y usando el método push, agrega los siguientes objetos al arreglo: {raza: "Pastor Aleman", color: "marron y negro" },{raza: "Beagle", color: "blanco, marrón y negro" },{auto: "Afgano", color: "negro" }.

```
let perros = [{raza: "Yorkshire Terrier", color: "marrón, negro y gris"
},{raza: "Shar Pei", color: "marrón" },{raza: "Schnauzer gigante",
color: "negro" }];
```

Método pop

Este método se utiliza para eliminar el último dato que tenemos en el arreglo, como se muestra en la siguiente imagen:

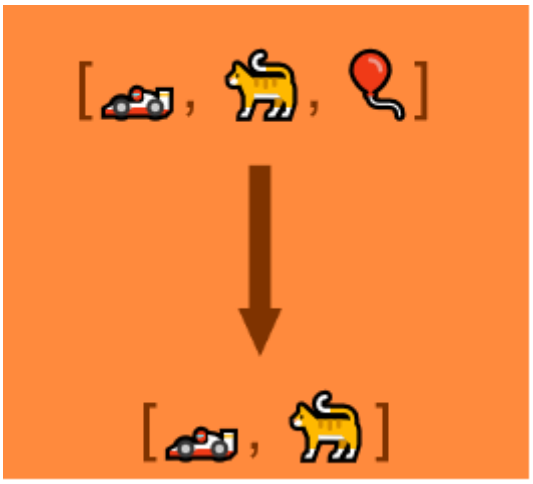

Imagen 3. Diagrama de ejemplificación del método pop().
Fuente: Desafío Latam

Veamos un ejemplo de como usar el método pop:

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

Lo anterior, nos entregará una salida como la siguiente:

```
(3) ["Erick", "Cristian", "Max"]
```

Ejercicio guiado: Método pop

Eliminar el último objeto del listado mostrado a continuación: [{titulo: "Harry Potter", pag: "300" },{titulo: "El principito", pag: "100" },{titulo: "De animales a dioses", pag: "350" }].

- ***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, agregar el siguiente código. En este caso, como estamos en presencia de un objeto dentro de un arreglo, utilizamos el método `pop()` sin pasar ningún tipo de valor entre los paréntesis:

```
let libros = [{titulo: "Harry Potter", pag: "300" },{titulo: "El principito", pag: "100" },{titulo: "De animales a dioses", pag: "350" }];
libros.pop();
console.log(libros);
```

- **Paso 3:** Al ejecutar el código anterior y revisar la consola del navegador web, nos generará una salida como la siguiente:

```
(2) [{titulo: "Harry Potter", paginas: "300" },{titulo: "El principito", paginas: "100" }]
```

Para mayores detalles y especificaciones **del método pop**, puede visitar el siguiente enlace

Ejercicio propuesto (2)

En base al arreglo presentado a continuación, elimina el último elemento del array llamado “El prisionero de azkaban”.

```
let librosJJK = ["La comunidad del anillo", "Las dos torres","El regreso del rey", "El prisionero de azkaban"];
```

Método shift

Shift al igual que el método anterior, nos permite eliminar un elemento de un arreglo, pero en este caso el elemento eliminado no es el último sino el primero, como se muestra en la siguiente imagen:

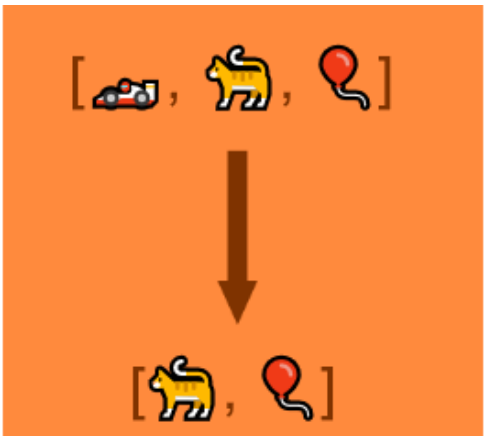

Imagen 4. Diagrama de ejemplificación del método shift().
Fuente: Desafío Latam

Veamos cómo funciona el método shift en el siguiente array.

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

Lo anterior, nos generará la siguiente salida:

```
(3) ["Cristian", "Max", "Claudia"]
```

Ejercicio guiado: Método shift

Emplear el método shift y eliminar el primer elemento del siguiente arreglo: {titulo: "Harry Potter", pag: "300" },{titulo: "El principito", pag: "100" },{titulo: "De animales a dioses", color: "350" }, por consiguiente:

- ***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*:** Si queremos eliminar el primer objeto de un array implementado `shift()`, debemos aplicar ese método al arreglo sin pasar ningún valor dentro de los paréntesis, quedando el siguiente código que agregamos al archivo script.js:

```
var libros = [{titulo: "Harry Potter", pag: "300" },{titulo: "El
principito", pag: "100" },{titulo: "De animales a dioses", pag: "350"
}];
libros.shift();
console.log(libros);
```

- ***Paso 3*:** Ejecutar en el navegador web lo anterior, se generará a siguiente salida:

```
(2) [{titulo: "El principito", paginas: "100" },{titulo: "De animales a
dioses", paginas: "350" }]
```

Para mayores detalles y especificaciones **del método shift**, puede visitar el siguiente enlace.

Ejercicio propuesto (3)

Escribe un array de cuatro (4) objetos con los siguientes datos: {cuerda: "Guitarra", cantidad: 7 },{viento: "Flauta", cantidad: 20 },{cuerda: "Violín", cantidad: 5 }, luego, quítale el primer elemento utilizando el método shift.

Métodos transformadores para modificar elementos

Competencias

- Reconocer los métodos integrados que modifican objetos o arreglos, para utilizar adecuadamente las características del lenguaje JavaScript.
- Desarrollar algoritmos utilizando métodos que permitan modificar los elementos de un arreglo acorde al lenguaje JavaScript para resolver un problema.

Introducción

Como vimos en el capítulo anterior, existen métodos que nos facilitan el trabajo con arreglos, para tareas recurrentes como agregar y eliminar elementos. A continuación, veremos cómo modificar elementos de uno o más arreglos, utilizando álgebra de conjuntos para dividir, unir o filtrar los datos en base a criterios que nos permitan resolver un problema dado.

A través de estos métodos, podremos continuar ejercitando el trabajo con arreglos y objetos en JavaScript y conoceremos más de las posibilidades que nos ofrece este lenguaje para trabajar con colecciones de datos.

Método split

El método split nos permite dividir una cadena de texto string en un arreglo. Aunque este método no se aplica en arreglos y objetos directamente, sí nos entrega como resultado un arreglo, como se muestra en la siguiente imagen:

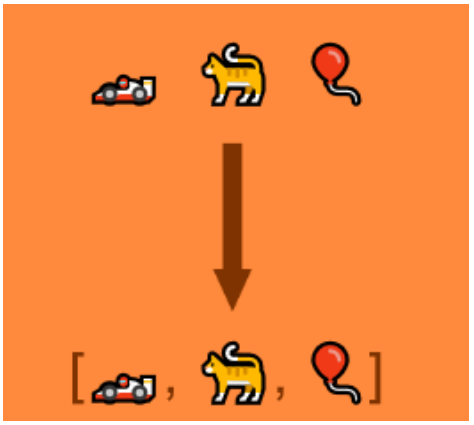

Imagen 5. Diagrama de ejemplificación del método split().
Fuente: Desafío Latam

Si quisiéramos dividir un texto que está separado por punto y coma, tal como se presentan los archivos con la extensión .csv, podríamos utilizar este método.

```
var cliente = 'Juan Carlos;29;jcarlos@gmail.com';
var arregloCliente = cliente.split(';');
console.log(arregloCliente);
```

El resultado de lo anterior sería un arreglo como el siguiente:

```
(3) ["Juan Carlos", "29", "jcarlos@gmail.com"]
```

Ejercicio guiado: Método split

Tenemos el siguiente texto en una variable denominada `filtros`: “Comida china RM nunoa” y queremos construir un arreglo con esa cadena de caracteres mediante los espacios que tiene. Sigamos los siguientes pasos:

- ***Paso 1*:** Crea 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, como queremos construir un nuevo arreglo con la variable `filtros`, utilizamos el método split. Dentro de los paréntesis pasamos un espacio como

parámetro, que será el separador común que utilizaremos para construir el nuevo arreglo, quedando el siguiente código que incluiremos en el archivo script.js:

```
var filtros = 'Comida china RM nunoa';
var filtros = filtros.split(' ');
console.log(filtros);
```

- ***Paso 3:*** El resultado de lo anterior sería un arreglo como el siguiente:

```
(3) ["Comida", "china", "RM", "nunoa"]
```

Para mayores detalles **del método split**, puede visitar el siguiente enlace

Ejercicio propuesto (4)

¿Cuál es la cadena de texto que al pasar por el método split da como resultado este array `["mesa","silla","comedor"]`? No puedes usar ',' para conectar la cadena de texto, valida con el método split.

Método join

Imagina que estás desarrollando una página y necesitas imprimir los datos almacenados en un arreglo dándoles formato, como por ejemplo separados por un guión. El método join une todos los elementos de un arreglo especificado en una sola cadena de texto; este método tiene un comportamiento parecido al método "toString()", pero la diferencia es que se puede especificar el separador entre cada elemento del arreglo, como se muestra en la imagen 6:

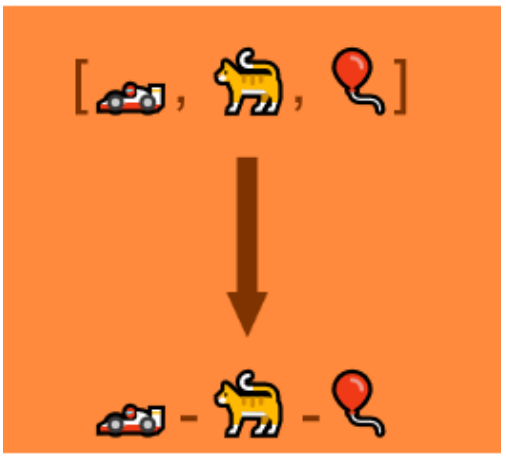

Imagen 6. Diagrama de ejemplificación del método join().
Fuente: Desafío Latam

Demos formato a este array transformándolo a una cadena de texto separada por "-"

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

Lo anterior, nos entregará una salida como la siguiente:

```
Erick - Cristian - Max - Claudia
```

Ejercicio guiado: Método join

Se solicita dar formato al siguiente arreglo `["Rebeca Matte 18","Santiago"]` que contiene dos elementos como string e implementado el método join, donde el separador debe ser una coma (,). Para realizar esto, sigamos los siguientes pasos:

- ***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*:** Teniendo el arreglo definido, aplicar el método join y dentro del paréntesis agregamos una coma con un solo espacio, para poder separar los dos elementos del arreglo en uno solo pero separado por la coma. Quedando el siguiente código que agregamos al archivo script.js:

```
var dirección = ["Rebeca Matte 18","Santiago"];
console.log(dirección.join(", "));
```

- ***Paso 3:*** Ejecutar el código anterior en el navegador web, nos entregará una salida como la siguiente:

```
"Rebeca Matte 18, Santiago"
```

Para mayores detalles y especificaciones **del método join**, puede visitar el siguiente enlace

Ejercicio propuesto (5)

Estás consumiendo una interfaz de información que devuelve un listado de pokemones: `["pikachu","ditto","bulbasaur"]`, pero los propietarios de la página web donde estas realizando el trabajo necesitan que el despliegue de la información sea en el título de la

página web, por lo que necesitas transformar en texto para que quede separado por guiones (-) implementado el método join.

Método map

El método map nos permite obtener un array procesado en base a otro array, como muestra la siguiente imagen:

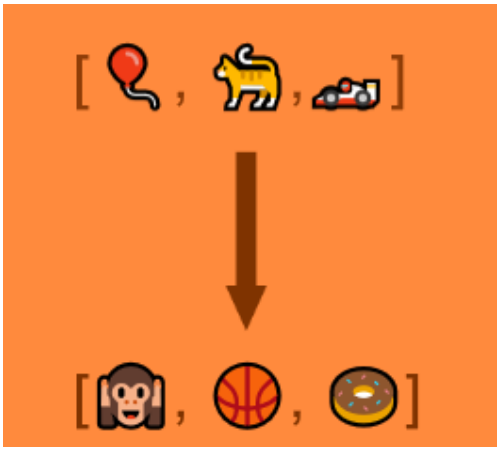

Imagen 7. Diagrama de ejemplificación del método map().
Fuente: Desafío Latam

Por consiguiente, la estructura básica es:

```
var nuevo_array = arreglo.map(function (elemento, index, array) {
      // Elemento devuelto de nuevo_array
};
```

Ejercicio guiado: Método map

Partiendo de un arreglo que posee distintos objetos, se requiere crear un nuevo arreglo que contenga toda la información de los clientes originales y que precise un nuevo elemento indicando si el cliente es mayor de edad. El arreglo original es:

```
var clientes = [
  {nombre: 'Juan', edad: 28},
  {nombre: 'Carlos', edad: 17},
  {nombre: 'Karla', edad: 27},
];
```

- ***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 escribir el arreglo facilitado al inicio del ejercicio así como la estructura básica del método map, el cual lleva una función interna encargada de ejecutarse por cada elemento que contenga el arreglo al cual le estamos aplicando el método:

```
clientes_modificado = clientes.map(function(cliente) {
});
```

- ***Paso 3*:** Al igual que el método **map** realiza una iteración sobre cada elemento de un arreglo, en este caso, el arreglo de clientes. Ahora por cada cliente en el array realizamos una validación (utilizando if) para revisar si la edad es mayor o igual a 18 (para verificar que sea mayor de edad). En el caso de ser verdadero, agregamos una propiedad llamada adulto con el valor true. Caso contrario, agregamos igual la propiedad adulto, pero con valor false, luego retornamos el objeto. Finalmente, implementamos un console.log para mostrar el arreglo final:

```
clientes_modificado = clientes.map(function(cliente) {
   if (cliente.edad >= 18) {
       cliente.adulto = true;
  } else {
       cliente.adulto = false;
  }
   return cliente;
});

console.log(clientes_modificado);
```

- ***Paso 4:*** Ejecutar el código anterior el resultado será:

```
0: {nombre: "Juan", edad: 28, adulto: true}
1: {nombre: "Carlos", edad: 17, adulto: false}
2: {nombre: "Karla", edad: 27, adulto: true}
```

Como se puede ver, el array incorpora la nueva propiedad que indica si el cliente en cuestión es adulto o no. Como dato, es importante recordar retornar el objeto que estemos procesando, de lo contrario se generará un array con valores undefined (a menos que esa sea la intención).

Ejercicio guiado: Multiplicando elementos con map

Aplicar el método map para multiplicar por 2 los elementos del siguiente arreglo [10,15,20,25,30] y guardar el nuevo arreglo en una variable.

- ***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 escribir el arreglo facilitado al inicio del ejercicio así como la estructura básica del método map, que lleva una función interna encargada de ejecutarse por cada elemento que contenga el arreglo, luego, dentro de la función, lo que se quiere es retornar el valor del elemento multiplicado por dos.

```
var numeros = [10,15,20,25,30];

numeros_nuevo = numeros.map(function(num) {
   return num * 2;
});
console.log(numeros_nuevo);
```

- ***Paso 3*:** El resultado del código anterior debería ser:

```
(5) [ 20, 30, 40, 50, 60 ]
```

Para más información y detalles sobre **el método map,** puedes visitar el siguiente enlace.

Ejercicio propuesto (6)

Utilizando el método map, crea un nuevo arreglo con el cuadrado de cada elemento del arreglo original. Puedes utilizar el método Math.pow() .

```
var numeros = [1,2,3,5,8,13,21];
```

Método reduce

Con reduce podemos transformar los elementos de un array en un valor único. Es de gran utilidad si por ejemplo, deseamos sumar dichos valores.

La sintaxis básica de este método, es la siguiente:

```
arreglo.reduce(callback(acumulador, valorActual[, índice[, array]])[,
valorInicial])
```

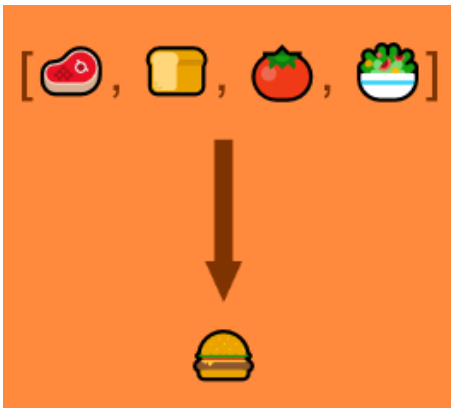

Imagen 8. Diagrama de ejemplificación del método reduce().
Fuente: Desafío Latam

Por ejemplo, si queremos implementar el método reduce para agrupar un arreglo con varios caracteres en una sola palabra:

```
['H','o','l','a'].reduce(function(valorAnterior, valorActual, indice,
vector){
  return valorAnterior + valorActual;
});

// Primera llamada
valorAnterior = H, valorActual = o, indice = 1

// Segunda llamada
valorAnterior = Ho, valorActual = l, indice = 2

// Tercera llamada
valorAnterior = Hol, valorActual = a, indice = 3

// Valor Devuelto: “Hola”
```

Ejercicio guiado: Método reduce

Utilizar el método reduce para sumar las deudas de una persona, las cuales están almacenadas en un array, por ejemplo: [10000, 5000, 50000, 35000], podríamos hacerlo de la siguiente forma:

- ***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 escribir el arreglo facilitado al inicio del ejercicio con una variable "deudas", así como la estructura básica del método reduce. En este caso, el método reduce recibirá como parámetro una función, que a su vez recibirá dos parámetros: un contador y la deuda (los nombres pueden ser cualquiera según las necesidades del programador). El contador representa el resultado de lo ejecutado dentro de la función (valor anterior o acumulador), mientras que la deuda representa el elemento actual (valor actual) en el cual se está iterando.

```
var deudas = [10000, 5000, 50000, 35000];

var sumatoriaDeudas = deudas.reduce(function(contador, deuda){
   return contador + deuda;
});

console.log(sumatoriaDeudas);
```

- ***Paso 3:*** El resultado de lo anterior sería la suma de todos los elementos:

```
100000
```

Ejercicio guiado: Utilizando reduce sobre objetos

Es posible utilizar **reduce** sobre objetos, pero la ejecución cambia un poco. Se solicita imprimir por consola una lista de nombres de un arreglo sobre un único string, siendo los objetos del arreglo: {nombre: 'Juan', edad: 28}, {nombre: 'Carlos', edad: 17}, {nombre: 'Karla', edad: 27}, por lo que podríamos realizar lo siguiente:

- ***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, escribir el arreglo facilitado al inicio del ejercicio con una variable "deudas", así como la estructura básica del método reduce. En este caso, el método reduce recibirá un parámetro adicional (valor inicial) que representa el objeto o variable inicial sobre la cual se acumularán las operaciones realizadas dentro de reduce, la cual al finalizar la ejecución será retornada a la variable nombres y tiene como valor inicial un string vacío (" "). Dentro de la función entregada a reduce, acumulamos los nombres de cada cliente en el arreglo de clientes.

```
var clientes = [{nombre: 'Juan', edad: 28},{nombre: 'Carlos', edad:
17},{nombre: 'Karla', edad: 27}];

var nombres = clientes.reduce(function(acumulador, cliente){
       return acumulador + ' | ' + cliente.nombre;
}, '');

console.log(nombres);
```

- ***Paso 3:*** El resultado de la ejecución anterior es el siguiente:

```
| Juan | Carlos | Karla
```

Ejercicio guiado: Utilizando reduce para obtener totales

Del siguiente arreglo de objetos, se debe obtener la cantidad total de años de experiencia:

```
var experiencias = [
        {
           titulo: "Practica",
           anos: 1,
        },
        {
           titulo: "Programador Junior",
           anos: 2,
        },
        {
           titulo: "Programador Senior",
           anos: 4,
        },
        {
           titulo: "Jefe de proyecto",
           anos: 5,
        }
];
```

- ***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, trabajar con la estructura básica del método reduce, en este caso, recibirá un valor adicional. Este parámetro representa el objeto o variable

inicial que se le cargará al acumular como valor de inicio para la primera iteración, luego sumará lo que tenga en el acumulador con el valor actual que sería la experiencia en años de cada objeto, retornando un solo valor numérico final.

```
var anosDeExperiencia = experiencias.reduce(function(acumulador,
experiencia){
   return acumulador + experiencia.anos;
},0);

console.log(anosDeExperiencia);
```

- ***Paso 3:*** El resultado de lo anterior serían todos los años de experiencia.

```
12
```

Para más información sobre el **método reduce**, puedes visitar el siguiente enlace.

Ejercicio propuesto (7)

Obtener el promedio de notas del siguiente arreglo = [6,7,7,5,4] utilizando el método reduce.

Método filter

El método filter permite realizar un filtrado de objetos o elementos que se encuentran al interior de un arreglo y devolver el resultado en forma de arreglo mediante una iteración. Este método recibe como parámetro una función, la cual a su vez recibe como parámetro el elemento en cuestión sobre el cual se está iterando, como se muestra en la imagen 9:

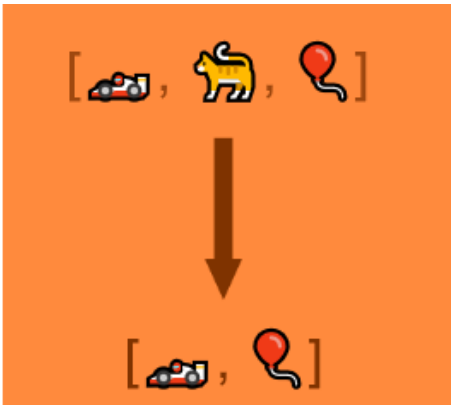

Imagen 9. Diagrama de ejemplificación del método filter().
Fuente: Desafío Latam

La sintaxis básica del método filter es la siguiente:

```
var nuevoArreglo = arreglo.filter(callback(currentValue[, index[,
array]])[, thisArg])
```

Ejercicio guiado: Método filter

Se desea filtrar y obtener solo los clientes que sean mayores de edad del siguiente array con datos de clientes:

```
var clientes = [
  {nombre: 'Juan', edad: 28},
  {nombre: 'Carlos', edad: 17},
  {nombre: 'Karla', edad: 27},
];
```

- ***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, agregar el arreglo llamado clientes, el cual contiene tres objetos que corresponden a clientes, de los cuales tan sólo uno es menor de edad. Luego, creamos una variable llamada adultos donde almacenaremos el resultado del filtro (personas mayores de edad). En el interior del método filter, retornamos un valor booleano para saber si el objeto se filtra o no. En este caso, el filtro es la edad del cliente: si la edad es mayor o igual a 18, el objeto no se filtrará, en cambio, si la edad del cliente es menor a eso, se filtrará y no se almacenará.

```
var clientes = [
  {nombre: 'Juan', edad: 28},
  {nombre: 'Carlos', edad: 17},
  {nombre: 'Karla', edad: 27},
];

var adultos = clientes.filter(function(cliente){
       return cliente.edad >= 18
});

console.log(adultos);
```

- ***Paso 3:*** Como resultado se imprimirá por consola lo siguiente:

```
{nombre: "Juan", edad: 28}
{nombre: "Karla", edad: 27}
```

Ejercicio guiado: Utilizando filter con una condición

Del siguiente arreglo, obtener sólo los trabajos donde se mantuvo más de 2 años continuos.

```
var experiencias = [
        {
           titulo: "Practica",
           anos: 1,
        },
        {
           titulo: "Programador Junior",
           anos: 2,
        },
        {
           titulo: "Programador Senior",
           anos: 4,
        },
        {
           titulo: "Jefe de proyecto",
           anos: 5,
        }
];
```

- ***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, agregar el arreglo indicado en el enunciado del ejercicio, el cual contiene cuatro (4) elementos con una información en específico, de la que solo nos interesa por el momento los años, porque por medio de esa información se aplicará el filtro correspondiente, en este caso, que tenga más de dos años de experiencia continua. Luego, creamos una variable llamada `anosDeExperiencia` donde almacenaremos el resultado del filtro, es decir, solo los que tengan más de dos años de experiencia serán guardados en la nueva variable.

```
var anosDeExperiencia = experiencias.filter(function(experiencia){
   return experiencia.anos > 2;
});
console.log(anosDeExperiencia);
```

- ***Paso 3:*** El resultado es:

```
(2) [{ titulo: "Programador Senior", anos: 4 },{ titulo: "Jefe de
proyecto", anos: 5 }]
```

Si deseas obtener más información acerca del **método filter,** puedes visitar el siguiente enlace

Ejercicio propuesto (8)

Del siguiente arreglo obtenga los pokemones con experiencia (level) mayor a 30.

```
var pokemones = [
  {nombre: 'Pikachu', level: 28},
  {nombre: 'Charmander', level: 22},
  {nombre: 'Dito', level: 37},
];
```

Método concat

Mediante el método concat, podemos fusionar los elementos de dos o más arrays dentro de un solo resultado. Puede ser muy útil para unir elementos que pertenezcan a una misma clasificación, como se muestra en la siguiente imagen:

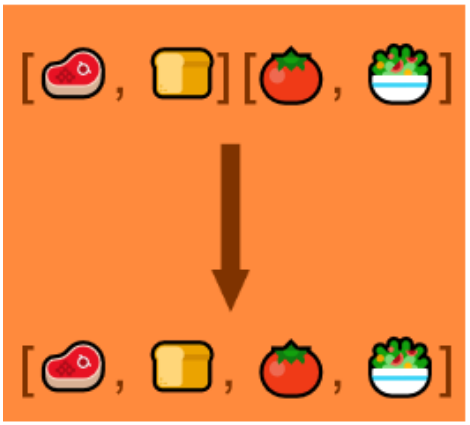

Imagen 10. Diagrama de ejemplificación del método concat().
Fuente: Desafío Latam

La sintaxis básica del método concat es la siguiente:

```
var nuevo_arreglo = arreglo_1.concat(arreglo_2);
```

Ejercicio guiado: Método concat

Se necesita fusionar los elementos de dos arreglos denominados `autosCompactos` y `autosDeportivos`, de acuerdo a lo siguiente:

```
var autosCompactos = [
  {marca: 'Toyota', modelo: 'Corolla', combustible: 'Gasolina'},
  {marca: 'Mazda', modelo: '3', combustible: 'Gasolina'},
  {marca: 'Honda', modelo: 'Civic', combustible: 'Gasolina'},
  {marca: 'Bmw', modelo: '116d', combustible: 'Diesel'},
];
var autosDeportivos = [
  {marca: 'Opel', modelo: 'Astra OPC', combustible: 'Gasolina'},
  {marca: 'Renault', modelo: 'Megane RS', combustible: 'Gasolina'},
  {marca: 'Mitsubishi', modelo: 'Lancer Evo', combustible: 'Gasolina'},
];
```

- ***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, agregar los dos arreglos indicados, luego mediante el método concat al arreglo de `autosCompactos` le unimos con el arreglo de `autosDeportivos`.

```
var autos = autosCompactos.concat(autosDeportivos);

console.log(autos);
```

- ***Paso 3:*** El resultado de la operación anterior sería un arreglo con el siguiente contenido:

```
0: {marca: "Toyota", modelo: "Corolla", combustible: "Gasolina"}
1: {marca: "Mazda", modelo: "3", combustible: "Gasolina"}
2: {marca: "Honda", modelo: "Civic", combustible: "Gasolina"}
3: {marca: "Bmw", modelo: "116d", combustible: "Diesel"}
4: {marca: "Opel", modelo: "Astra OPC", combustible: "Gasolina"}
5: {marca: "Renault", modelo: "Megane RS", combustible: "Gasolina"}
```

```
6: {marca: "Mitsubishi", modelo: "Lancer Evo", combustible: "Gasolina"}
```

Ejercicio guiado: Utilizando concat para unir arreglos

Se requiere unir los siguientes arreglos de letras, letrasA = ["a","b","c","d"] y letrasB = ["e","f","g","h"], en un solo arreglo.

- ***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, agregar los dos arreglos indicados, luego mediante el método concat al arreglo letrasA le unimos el arreglo letrasB para obtener un solo arreglo.

```
var letrasA = ["a","b","c","d"];
var letrasB = ["e","f","g","h"];

var letras = letrasA.concat(letrasA);
console.log(letras);
```

- ***Paso 3:*** Al ejecutar el código anterior, el resultado es:

```
(8) ["a","b","c","d","e","f","g","h"];
```

Si deseas obtener más información acerca del **método concat**, puedes visitar el siguiente enlace.

Ejercicio propuesto (9)

Escribir el código para obtener este arreglo final usando el método concat `var resultado = ["perro","gato","ratón","iguana"]` en base a los siguientes arreglos:

```
var animalesA = ["perro","gato"];
var animalesB = ["ratón","iguana"];

var animales = animalesA.concat(animalesB);
console.log(animales);
```

Métodos para acceder a elementos

Competencias

- Reconocer los métodos integrados para acceder, recorrer y ordenar elementos de un objeto o arreglo, para utilizar adecuadamente las características del lenguaje JavaScript.

- Desarrollar algoritmos utilizando operaciones sobre los elementos de un arreglo, iteración, filtrado u ordenamiento de elementos acorde al lenguaje JavaScript para resolver un problema.

Introducción

Como pudimos observar, los métodos integrados nos facilitan el trabajo con colecciones de datos ya que permiten manipular las estructuras de acuerdo a las necesidades del problema.

Si bien, modificar elementos de una colección es una tarea relevante, también lo es obtener los datos ingresados, identificar si existen en un conjunto y recorrerlos con la finalidad de consultarlos o actualizar la información.

A continuación, veremos aquellos métodos integrados que nos permiten realizar estas tareas y comprender cuáles son las herramientas que nos provee JavaScript para resolver problemas relativos a estructuras de datos.

Método sort

Este método nos permite ordenar de manera alfabética los tipos de datos string. Para otros tipos de datos, sort ordena de acuerdo a su valor en el estándar Unicode:

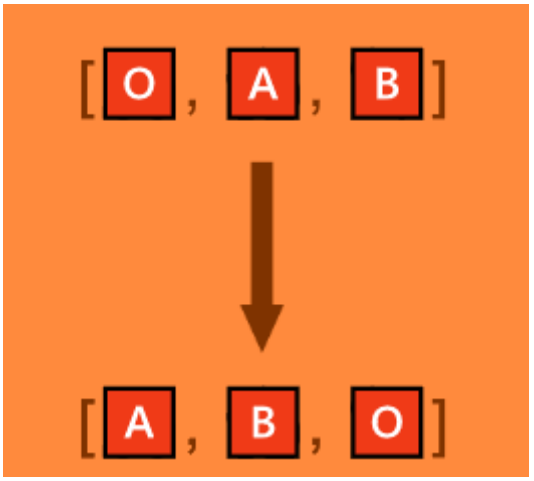

Imagen 11. Diagrama de ejemplificación del método sort().
Fuente: Desafío Latam

Ordenemos el siguiente array usando el método sort:

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

La salida para lo anterior sería:

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

Ejercicio guiado: Método sort

Se entrega un arreglo de números o con tipos de datos numéricos (number) y solicitan ordenar los números pero en base al primer dígito de menor a mayor. Siendo los elementos del arreglo: “1,5,20,23”. Para realizar esto, debemos seguir los siguientes pasos:

- ***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 de JavaScript, plantear el arreglo y crear una variable denominada num para guardar la información. Luego, mediante el uso de la consola

mostrar el resultado al aplicar el método sort al arreglo. Quedando de la siguiente manera:

```
var numeros = [1, 5, 20, 23];
console.log(numeros.sort());
```

- ***Paso 3*:** Ejecutar el código anterior, obtendremos la siguiente salida:

```
(4) [1, 20, 23, 5]
```

Como podemos ver, la salida en este caso sí está ordenada, pero de acuerdo a lo establecido por Unicode. Para mayores detalles y especificaciones **del método sort**, puede visitar el siguiente enlace.

Ejercicio propuesto (10)

¿Cuál es el resultado de aplicar sort al siguiente array?

```
var lista =["Pepe",5,2,"Diego","1"]
```

Método reverse

Con reverse, se puede invertir el orden de los elementos dentro de un array, como se muestra en la siguiente imagen:

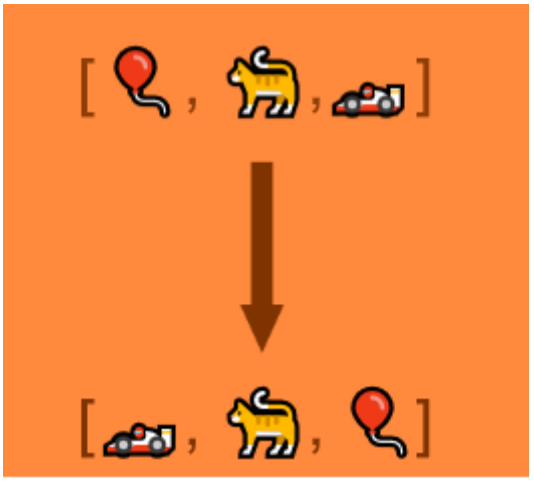

Imagen 12. Diagrama de ejemplificación del método reverse().
Fuente: Desafío Latam

Veamos cómo usar el método reverse:

```
var muchachos = ["Juan", "Lucas", "Pedro", "Marcos"];
console.log(muchachos.reverse());
```

El resultado de lo anterior sería lo siguiente:

```
(4) ["Marcos", "Pedro", "Lucas", "Juan"]
```

Ejercicio guiado: Método reverse

Aplicar el método reverse al siguiente arreglo con los elementos: "Charizard, Charmeleon, Charmander", logrando ordenar de forma inversa los datos que contiene el arreglo. Para realizar esto, sigamos los siguientes pasos:

- ***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 de JavaScript, plantear el arreglo y crear una variable denominada pokémon para guardar la información sobre los pokemones. Ahora implementar el método reverse para darle un orden inverso a como están presentados actualmente los datos en el arreglo.

```
var pokemon = ["Charizard","Charmeleon","Charmander"];
console.log(pokemon.reverse());
```

- ***Paso 3:*** El resultado de lo anterior sería lo siguiente:

```
(3) [ "Charmander", "Charmeleon", "Charizard" ]
```

Para mayores detalles y especificaciones **del método reverse**, puede visitar el siguiente enlace.

Ejercicio propuesto (11)

Sin probar el código. ¿Cuál es el resultado de esta operación?

```
["perro", "gato", "ratón"].reverse()
```

Método forEach

Con el método forEach podemos recorrer cada elemento de un array y realizar alguna acción con respecto a cada uno de éstos. Es equivalente a recorrerlo con un ciclo for, salvo que este método se basa en cada elemento del array, por lo que no es necesario declarar variables a modo de índice ni realizar validaciones asociadas.

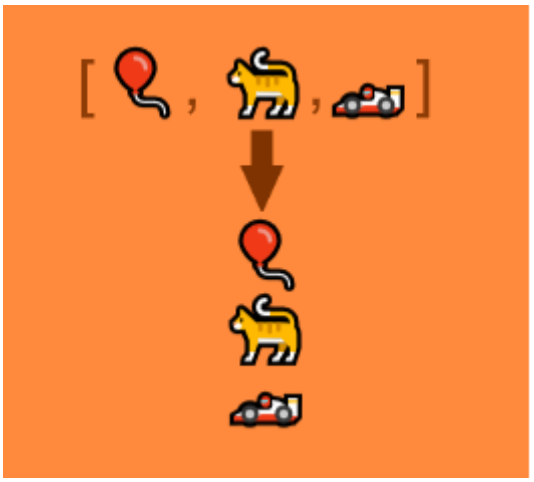

Imagen 13. Diagrama de ejemplificación del método forEach().
Fuente: Desafío Latam

Por consiguiente, la estructura básica del forEach está compuesta por una función que contendrá cada elemento del arreglo a medida que el ciclo se vaya cumpliendo.

```
arreglo.forEach(function callback(elemento, index, arreglo) {
      // cuerpo de la función dentro del iterado forEach
};
```

Ahora veamos un ejemplo con un arreglo de números:

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

arregloNumeros.forEach(function(num) {
   console.log(num);
});
```

El resultado es:

```
10
9
```

```
8
7
6
5
4
3
2
1
```

Ejercicio guiado: Método forEach

Recorrer un arreglo conformado por varios objetos y mostrar solamente el valor del elemento "nombre" que tiene cada uno de los objetos usando el método forEach. Siendo el arreglo:

```
let clientes = [
  {nombre: 'Juan', edad: 28},
  {nombre: 'Carlos', edad: 22},
  {nombre: 'Karla', edad: 27},
];
```

- ***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 este caso, trabajar con el arreglo denominado clientes que contiene tres (3) objetos en su interior, los cuales cada uno tiene como propiedad un nombre y la edad. Luego, accede al método forEach en el array clientes, el cual recibe como parámetro una función que a su vez tiene como parámetro una variable llamada cliente. Esta variable cliente representa cada uno de los elementos recorridos en cada iteración realizada por forEach. Luego, en el interior de la función, imprimimos por consola el nombre (propiedad que agregamos en el objeto) del objeto persona en el cual estamos iterando.

```
var clientes = [
  {nombre: 'Juan', edad: 28},
  {nombre: 'Carlos', edad: 22},
  {nombre: 'Karla', edad: 27},
];

clientes.forEach(function(cliente) {
   console.log(cliente.nombre);
```

```
});
```

- ***Paso 3:*** El resultado de la ejecución anterior sería como sigue:

```
Juan
Carlos
Karla
```

Para más información y detalles **del método forEach**, puedes visitar el siguiente enlace.

Ejercicio propuesto (12)

Para el arreglo mostrado a continuación, invierte e itera con un forEach los elementos dispuesto en el arreglo. Luego muestra el resultado final por medio de un console.log.

```
var nombres =
['Juan','Manuel','Elio','Ali','Yecelis','Yecenia','Laura'];
```

Método find

El método find nos permite obtener un objeto que cumpla alguna condición que especifiquemos. Funciona de forma muy similar al método filter, salvo que find retorna sólo un objeto y aquel que cumpla en primera instancia con la condición especificada. Es decir, retorna el valor del primer elemento del arreglo que cumpla la condición de prueba proporcionada.

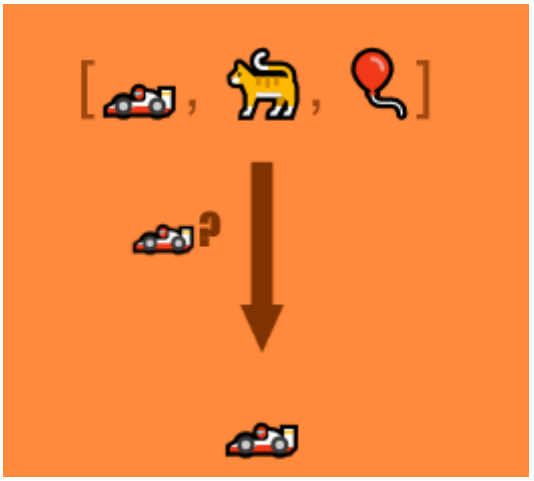

Imagen 14. Diagrama de ejemplificación del método find().
Fuente: Desafío Latam

La sintaxis básica de este método es:

```
arreglo.find(callback(element[, index[, array]])[, thisArg])
```

Ejercicio guiado: Método find

Para trabajar y practicar la implementación de este método, obtengamos de una lista de productos algún objeto que tenga el nombre kapo implementando el método find. El arreglo con productos es el siguiente:

```
var productos = [
  {nombre: 'coca-cola', precio: 990},
  {nombre: 'papas fritas', precio: 590},
  {nombre: 'ramitas', precio: 290},
  {nombre: 'kapo', precio: 190},
];
```

- ***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, agregar el arreglo indicado en el enunciado del ejercicio, el cual contiene cuatro (4) elementos con una información en específico de cada producto, como el nombre y el precio. En este caso, es de relevancia la información del nombre, debido a que la búsqueda indicada en el enunciado del ejercicio es con respecto al producto que lleva por nombre "kapo". Por lo tanto, el **método find** actuará como una rutina de iteración sobre cada elemento presente en el arreglo productos. Si encuentra una coincidencia que cumpla con la condición definida en su interior (producto.nombre == 'kapo'), se retorna el objeto en cuestión y termina su ejecución.

```
var productos = [
  {nombre: 'coca-cola', precio: 990},
  {nombre: 'papas fritas', precio: 590},
  {nombre: 'ramitas', precio: 290},
  {nombre: 'kapo', precio: 190},
];

var kapo = productos.find(function(producto){
   return producto.nombre == 'kapo'
});
```

```
console.log(kapo);
```

- ***Paso 3:*** Posteriormente, imprimir el objeto por consola y obtendremos la siguiente salida:

```
{nombre: "kapo", precio: 190}
```

- ***Paso 4:*** Obtener el producto que cuesta 290 pesos utilizando el método find del mismo arreglo de productos con el que venimos trabajando hasta el momento.

```
var productos = [
  {nombre: 'coca-cola', precio: 990},
  {nombre: 'papas fritas', precio: 590},
  {nombre: 'ramitas', precio: 290},
  {nombre: 'kapo', precio: 190},
];

var kapo = productos.find(function(producto){
   return producto.precio == 290
});
console.log(kapo);
```

- ***Paso 5:*** Al ejecutar el código anterior, el resultado es:

```
{nombre: "ramitas", precio: 290}
```

Si deseas obtener más información acerca del **método filter**, puedes visitar el siguiente enlace.

Ejercicio propuesto (13)

Del siguiente arreglo de objetos, busque el pokemon con el nombre Pikachu y muestre el resultado.

```
var pokemones = [
  {nombre: 'Charmander', level: 22},
  {nombre: 'Pikachu', level: 28},
  {nombre: 'Dito', level: 37},
];
```

Método findIndex

El método findIndex nos permite obtener el índice del objeto que cumpla en primera instancia con alguna condición que declaremos. Actúa de manera similar al método find, salvo que el resultado es distinto.

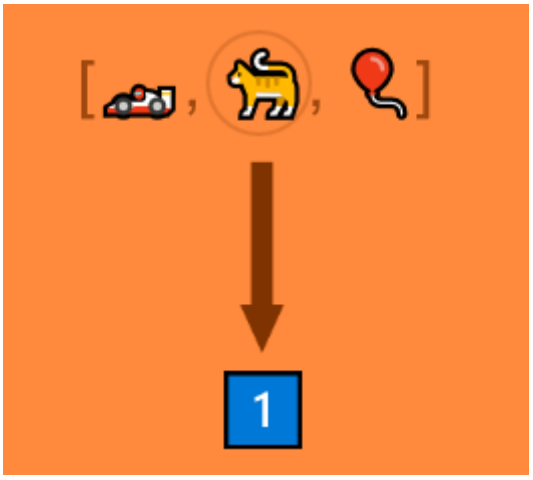

Imagen 15. Diagrama de ejemplificación del método findIndex().
Fuente: Desafío Latam

La sintaxis básica del método findIndex es:

```
arreglo.findIndex(callback( element[, index[, array]] )[, thisArg])
```

Ejercicio guiado: Método findIndex

El objetivo es recuperar el índice del objeto ramitas, en el arreglo utilizado anteriormente, realizaremos lo siguiente:

```
var productos = [
  {nombre: 'coca-cola', precio: 990},
  {nombre: 'papas fritas', precio: 590},
  {nombre: 'ramitas', precio: 290},
  {nombre: 'kapo', precio: 190},
];
```

- ***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, agregar el arreglo, el cual contiene cuatro (4) elementos con información en específico de cada producto, como el nombre y el precio, en este caso, es de relevancia la información del nombre, debido a que la búsqueda indicada en el enunciado del ejercicio, que se debe encontrar el indice si existe un producto con el nombre "ramitas". Por lo tanto, el **método findIndex** actuará como una rutina de iteración sobre cada elemento presente en el arreglo productos. Si encuentra una coincidencia que cumpla con la condición definida en su interior (producto.nombre == 'ramitas'), retorna el índice donde se encuentre el producto en cuestión y termina su ejecución.

```
var ramitasIndice = productos.findIndex(function(producto){
   return producto.nombre == 'ramitas'
});

console.log(ramitasIndice);
```

- ***Paso 3:*** El resultado de lo anterior sería el siguiente:

```
2
```

Ejercicio guiado: Utilizando findIndex para buscar la posición de un elemento

Buscar la posición en que se encuentra juan dentro del arreglo denominado personas, con los elementos: "pedro, juan, diego".

- ***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, agregar el arreglo correspondiente, el cual, se debe iterar para buscar el elemento llamado "juan" y retornar a la posición donde se encuentre. Por lo tanto, el método findIndex retornará el índice donde se encuentre el nombre en cuestión en el caso de encontrarlo y terminará su ejecución.

```
var personas = ["pedro","juan","diego"];

var juanIndice = personas.findIndex(function(persona){
   return persona == 'juan'
});

console.log(persona);
```

- ***Paso 3:*** Al ejecutar el código anterior, el resultado es:

```
1
```

Si deseas obtener más información acerca del **método findIndex**, puedes visitar el siguiente enlace.

Ejercicio propuesto (14)

Para el siguiente arreglo denominado experiencias, muestra la posición (índice) que contenga la experiencia mayor o igual a 5 años.

```
var experiencias = [
        {
           titulo: "Practica",
           anos: 1,
        },
        {
           titulo: "Programador Junior",
           anos: 2,
        },
        {
           titulo: "Programador Senior",
           anos: 4,
        },
        {
           titulo: "Jefe de proyecto",
           anos: 5,
        }
  ];
```

Método some

El método some nos permite verificar si algún objeto o elemento dentro de un array cumple con alguna condición que nosotros queramos verificar, es decir, revisar si un arreglo contiene o no un valor.

Imagen 16. Diagrama de ejemplificación del método some().
Fuente: Desafío Latam

La sintaxis básica del método some es:

```
arreglo.some(callback(element[, index[, array]])[, thisArg])
```

Ejercicio guiado: Método some

Implementar el método some para saber si existe un objeto del arreglo autos, que contenga algún automóvil que utilice como combustible el Diesel. El arreglo auto será el siguiente:

```
var autos = [
  {marca: 'Toyota', modelo: 'Corolla', combustible: 'Gasolina'},
  {marca: 'Mazda', modelo: '3', combustible: 'Gasolina'},
  {marca: 'Honda', modelo: 'Civic', combustible: 'Gasolina'},
  {marca: 'Bmw', modelo: '116d', combustible: 'Diesel'},
];
```

- ***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, agregar el arreglo correspondiente de autos indicado en el enunciado del ejercicio, el cual, se debe iterar con el método some para buscar e indicar si existe algún elemento que contenga que utilice como combustible el Diesel, retornando true o false si lo encuentra o no respectivamente.

```
var autos = [
  {marca: 'Toyota', modelo: 'Corolla', combustible: 'Gasolina'},
  {marca: 'Mazda', modelo: '3', combustible: 'Gasolina'},
  {marca: 'Honda', modelo: 'Civic', combustible: 'Gasolina'},
  {marca: 'Bmw', modelo: '116d', combustible: 'Diesel'},
];

var algunDiesel = autos.some(function(auto){
   return auto.combustible == 'Diesel'
});

console.log(algunDiesel);
```

- ***Paso 3:*** El resultado de la operación al ser ejecutado en el navegador web será **true**, ya que el automóvil de marca Bmw cumple la premisa establecida en el paso anterior.

```
true
```

Este método puede parecer muy similar al método find. De hecho, es común ver validaciones mediante find, pero el método some es mucho más correcto de emplear (semánticamente) para realizar validaciones booleanas (verdadero o falso).

Ejercicio guiado: Utilizando some para encontrar un número

Verificar si existe algún número menor de tres en el arreglo denominado números, con los elementos: "1,2,3,4,5,6,7". Resolvamos el ejercicio:

- ***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, agregar el arreglo correspondiente con los números indicados, luego, mediante el método some, apliquemos la función con un retorno solo para números menores que 3.

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

var algunMenorTres = numeros.some(function(num){
   return num < 3
});
console.log(algunMenorTres);
```

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

```
true
```

Si deseas obtener más información acerca del **método some**, puedes visitar el siguiente enlace.

Ejercicio propuesto (15)

¿Existe en el listado algún pokémon tipo agua?. Implementa el método some para verificar la condición indicada, retornando true o false.

```
var pokemones = [
  {nombre: 'Charmander', tipo: "Fuego"},
  {nombre: 'Pikachu', tipo: "Electricidad"},
  {nombre: 'Dito', tipo: "normal"},
];
```

Resumen

En esta lectura hemos podido comprender y experimentar los distintos métodos predefinidos que provee JavaScript para el manejo de arrays.

Vimos que existen métodos que nos permiten manipular el contenido de los arreglos: agregar, eliminar y modificar datos, además de crear arreglos en base a criterios definidos e información de otras fuentes.

También aprendimos que existen métodos que nos permiten acceder fácilmente a la información contenida en las colecciones de datos: consultar, ordenar y conocer si un valor está presente.

Estas herramientas son muy poderosas para resolver distintos tipos de problemas cotidianos y nos servirán como base para resolver desafíos en nuestro trabajo diario como programadores.

Solución de los ejercicios propuestos

1. Partiendo el arreglo mostrado a continuación, y usando el método push, agrega los siguientes objetos al arreglo: {raza: "Pastor Aleman", color: "marron y negro" },{raza: "Beagle", color: "blanco, marrón y negro" },{auto: "Afgano", color: "negro" }.

```
let perros = [{raza: "Yorkshire Terrier", color: "marrón, negro y gris"
},{raza: "Shar Pei", color: "marrón" },{raza: "Schnauzer gigante",
color: "negro" }];
perros.push({raza: "Pastor Aleman", color: "marron y negro" },{raza:
"Beagle", color: "blanco, marrón y negro" },{auto: "Afgano", color:
"negro"});
console.log(perros);
```

2. Partiendo del arreglo mostrado a continuación, elimina el último elemento del array llamado “El prisionero de azkaban”.

```
let librosJJK = ["La comunidad del anillo", "Las dos torres","El regreso
del rey", "El prisionero de azkaban"];
librosJJK.pop();
console.log(librosJJK);
```

3. Escribe un array de cuatro (4) objetos con los siguientes datos: {cuerda: "Guitarra", cantidad: 7 },{viento: "Flauta", cantidad: 20 },{cuerda: "Violín", cantidad: 5 }, luego, quítale el primer elemento utilizando el método shift.

```
let instrumentos = [{cuerda: "Guitarra", cantidad: 7 },{viento:
"Flauta", cantidad: 20 },{cuerda: "Violín", cantidad: 5 }];
instrumentos.shift();
console.log(instrumentos);
```

4. ¿Cuál es la cadena de texto que al pasar por el método split da como resultado este array `["mesa","silla","comedor"]`? No puedes usar ',' para conectar la cadena de texto, valida con el método split.

```
var filtros = 'mesa silla comedor';
var filtros = filtros.split(' ');
console.log(filtros);
```

5. Estás consumiendo una interfaz de información que devuelve un listado de pokemones: `["pikachu","ditto","bulbasaur"]`, pero los propietarios de la página web donde estas realizando el trabajo necesitan que el despliegue de la información sea en el título de la página web, por lo que necesitas transformar en texto para que quede separado por guiones (-) implementado el método join.

```
var pokemons = ["pikachu","ditto","bulbasaur"];
console.log(pokemons.join(" - "));
```

6. Utilizando el método map, crea un nuevo arreglo con el cuadro de cada elemento del arreglo original. Puedes utilizar el método Math.pow().

```
var numeros = [1,2,3,5,8,13,21];

numeros_nuevo = numeros.map(function(num) {
   return Math.pow(num,2);
});
console.log(numeros_nuevo);
```

7. Obtenga el promedio de notas del siguiente arreglo de notas = [6,7,7,5,4] utilizando el método reduce.

```
var notas = [6, 7, 7, 5, 4];

var sumaNotas = notas.reduce(function(acumulador, nota){
   return (acumulador + nota);
},0);

var promedio = (sumaNotas / notas.length);

console.log(promedio)
```

8. Del siguiente arreglo denominado pokemones, obtenga los pokemones con experiencia (level) mayor a 30.

```
var pokemones = [
  {nombre: 'Pikachu', level: 28},
  {nombre: 'Charmander', level: 22},
  {nombre: 'Dito', level: 37},
];
```

```
var level = pokemones.filter(function(experiencia){
   return experiencia.level > 30;
});
console.log(level);
```

9. Escriba el código para obtener este arreglo final usando el método concat `var resultado = ["perro","gato","ratón","iguana"];`

```
var animalesA = ["perro","gato"];
var animalesB = ["ratón","iguana"];

var animales = animalesA.concat(animalesB);
console.log(animales);
```

10. ¿Cuál es el resultado de aplicar sort al siguiente array `var lista =["Pepe",5,2,"Diego","1"]`?

```
var lista = ["Pepe",5,2,"Diego","1"];
console.log(lista.sort()); // Array(5) [ "1", 2, 5, "Diego", "Pepe" ]
```

11. Sin probar el código. ¿Cuál es el resultado de esta operación `["perro", "gato", "ratón"].reverse()` ?

```
Array(3) [ "ratón", "gato", "perro" ]
```

12. Para el arreglo mostrado a continuación, invierte e itera con un forEach los elementos dispuesto en el arreglo. Luego muestra el resultado final por medio de un console.log.

```
var nombres =
['Juan','Manuel','Elio','Ali','Yecelis','Yecenia','Laura'];

nombres.reverse();
nombres.forEach(function(nombre) {
   console.log(nombre);
});
```

13. Del siguiente arreglo de objetos, busque el pokemon con el nombre Pikachu y muestre el resultado.

```
var pokemones = [
  {nombre: 'Charmander', level: 22},
  {nombre: 'Pikachu', level: 28},
  {nombre: 'Dito', level: 37},
];

var pikachu = pokemones.find(function(pokemon){
   return pokemon.nombre == 'Pikachu';
});
console.log(pikachu);
```

14. Para el siguiente arreglo denominado experiencias, muestra la posición (índice), que contenga la experiencia mayor o igual a 5 años.

```
var experiencias = [
        {
           titulo: "Practica",
           anos: 1,
        },
        {
           titulo: "Programador Junior",
           anos: 2,
        },
        {
           titulo: "Programador Senior",
           anos: 4,
        },
        {
           titulo: "Jefe de proyecto",
           anos: 5,
        }
  ];

var expeIndice = experiencias.findIndex(function(persona){
   return persona.anos >= 5;
});

console.log(expeIndice);
```

15. ¿Hay en el listado algún pokémon tipo agua? Implementa el método some para verificar la condición indicada, retornando true o false.

```
var pokemones = [
  {nombre: 'Charmander', tipo: "Fuego"},
  {nombre: 'Pikachu', tipo: "Electricidad"},
  {nombre: 'Dito', tipo: "normal"},
];

var tipoAgua = pokemones.some(function(tipo){
   return tipo.tipo == "agua"
});
console.log(tipoAgua);
```