educación IT
CODE YOUR FUTURE

JavaScript desde cero

Módulo 3

Arreglos (*arrays*)

Arreglos (*arrays*)

Los ***arrays***, (también llamados ***arreglos*** o ***colecciones***) permiten definir, bajo una misma estructura, una **serie de valores útiles** dentro de nuestra aplicación.

En JS, se utilizan dos tipos de *arrays*:

- *Arrays* de **elementos**.
- *Arrays* de **objetos**.

Si bien es posible crear *arrays* que almacenan diferentes tipos de datos, no es una práctica común, dado que le agrega complejidad a la aplicación, al momento de validar a qué tipo de dato corresponde cada valor almacenado.

Ejemplo: *array* de objetos

Ante la necesidad de crear un *array* que **almacene diferentes valores, en cada uno de sus índices**, lo más apropiado será estructurar un ***array de objetos***, que aporte orden y entendimiento.

Arrays

```
const personas = ['Fer', 'Nico', 'July', 'Laura', 'Greta'];

const pouporri = ['Moon', 21, 75, true, personas];
```

Cada valor almacenado en un *array*, obtiene una **posición numérica**, denominada ***índice***. Esta posición numérica inicia siempre a partir del **valor *0* (cero)**, **y se incrementa en un dígito por cada nuevo valor** contenido en el *array*.

Arrays

```
const personas = ['Fer', 'Nico', 'July', 'Laura', 'Greta'];
// posición          0       1        2        3        4
```

Para leer algún valor que esté almacenado en el *array*, simplemente **se define el nombre del *array*, y se encierra entre corchetes el valor numérico de la posición** que se desea leer:

Arrays

```
const personas = ['Fer', 'Nico', 'July', 'Laura', 'Greta'];
// posición          0       1        2        3        4


personas[3];     //devuelve 'Laura'
```

Ver contenido del *array* y depurarlo

Cuando trabajamos con *arrays*, la forma más efectiva de ver su contenido y depurarlo, es utilizar el **objeto `console`**, y su **método `.table()`**.

Este método representa un *array* en ***DevTools > Console***, que muestra en formato tabular, y representa sus índices en una segunda columna.

```
const paisesAmericaSur = ["Argentina", "Bolivia", "Brasil", "Chile", "Colombia",
"Ecuador", "Guyana", "Paraguay", "Perú", "Surinam", "Uruguay", "Venezuela"];

console.table(paisesAmericaSur);
```

(índ…	Valor
0	'Argentina'
1	'Bolivia'
2	'Brasil'
3	'Chile'
4	'Colombia'
5	'Ecuador'
6	'Guyana'
7	'Paraguay'
8	'Perú'
9	'Surinam'
10	'Uruguay'
11	'Venezuela'

Contabilizar el total de elementos almacenados

Si se necesita contabilizar el total de elementos que tiene almacenados, se puede recurrir a la **propiedad `.length`**.

Retornará un **valor numérico, correspondiente al total de elementos almacenados en el *array***.

```
const paisesAmericaSur = ["Argentina", "Bolivia", "Brasil", "Chile", "Colombia",
"Ecuador", "Guyana", "Paraguay", "Perú", "Surinam", "Uruguay", "Venezuela"];

console.log("Total de países en el arreglo:", paisesAmericaSur.length);

// IMPRIMIRÁ:

// Total de países en el arreglo: 12
```

Métodos de *arrays*

Además de la propiedad `.length`, los arreglos cuentan con una serie de **métodos que permiten operar con sus valores**. A través de ellos, es posible:

- Agregar o quitar elementos.
- Modificar elementos existentes.
- Cambiar elementos del arreglo.
- Ordenar de forma ascendente o descendente.
- Fusionar arreglos.
- Buscar, filtrar, reestructurar, calcular, validar.
- Crear nuevos arreglos a partir de uno existente.

Los métodos están en constante evolución; por lo tanto, siempre nos conviene seguir las tendencias del mercado para ver aquellos nuevos métodos que surgen y su impacto en el manejo de *arrays*.

educación IT
CODE YOUR FUTURE

Métodos de *arrays*

Métodos de *arrays*

Método	Descripción
`.push`	Agrega un nuevo elemento al final del *array*.
`.unshit`	Agrega un nuevo elemento al principio del *array*.
`.pop`	Quita el último elemento del *array*.
`.shift`	Quita el primer elemento del *array*.
`.slice`	Crea un nuevo *array* con elementos de otro *array*, de forma parcial o total.
`.splice`	Quita uno o más elementos del *array* de cualquier posición. También reemplaza uno o más elementos de un *array*.
`.indexOf`	Permite identificar si un elemento existe, o no, dentro del *array*. Retorna **`true`** o **`false`**, según el resultado.
`.includes`	Permite identificar si existe o no un elemento dentro de un *array*.

Más métodos de *arrays* que complementan la lista anterior:

Método	Descripción
`.concat`	Agrega un nuevo elemento al final del *array*.
`.join`	Agrega un nuevo elemento al principio del *array*.
`.sort`	Ordena de forma ascendente los elementos de un *array*.
`.reverse`	Revierte el orden de los elementos de un *array*.

Veamos en detalle cada uno de los métodos enlistados, en las siguientes diapositivas.

.push()

El método `.push()` **agrega un nuevo elemento al final del *array***. Este elemento se debe agregar **dentro de los paréntesis** del método.

Veamos el ejemplo de código:

```
const personas = ["Fer", "Nico", "July"];

personas.push("Laura");

// El array contendrá, ahora: Fer, Nico, July, Laura
```

.unshift()

El método `.unshift()` **agrega un nuevo elemento al inicio del *array***. Este elemento se debe agregar **dentro de los paréntesis** del método.

Veamos el ejemplo de código:

```
const personas = ["Fer", "Nico", "July", "Laura"];

personas.unshift("Greta");

// El array contendrá, ahora: Greta, Fer, Nico, July, Laura
```

.pop()

El método `.pop()` **quita el último elemento del *array***.

No es necesario definir ningún parámetro, ya que su acción es automática.

```
const personas = ["Greta", "Fer", "Nico", "July", "Laura"];

personas.pop();

// El array contendrá, ahora: Greta, Fer, Nico, July
```

.shift()

El método `.shift()` **quita el primer elemento del *array***.

No es necesario definir ningún parámetro, ya que su acción es automática.

```
const personas = ["Greta", "Fer", "Nico", "July"];

personas.shift();

// El array contendrá, ahora: Fer, Nico, July
```

.slice()

El método `.slice()` (del inglés *rebanar*) **crea un nuevo *array* con elementos de un *array* existente**.

Recibe dos parámetros:

1. El primero, corresponde al índice del *array* desde **donde inicia el proceso** mientras que,
2. el segundo parámetro, **hasta qué índice debe "rebanar"**.

Para tener en cuenta:

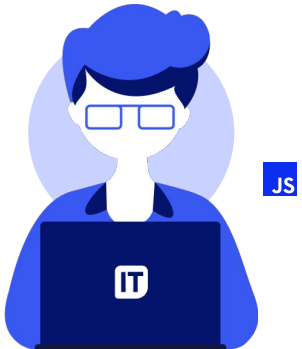
- El índice indicado en el segundo parámetro no está incluido en el *array* resultante.
- El *array* original no se altera. Los elementos seguirán estando allí.
- Para capturar los elementos que es necesario obtener, se debe declarar un nuevo *array*.

Ejemplo:

```
const personas = ["Greta", "Fer", "Nico", "July", "Laura"];

const hijos = personas.slice(2, 4);

// El array 'hijos' contendrá: Nico, July
```

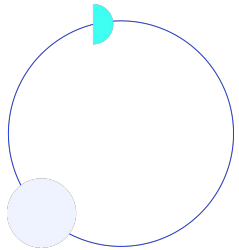


El método `.slice()` puede recibir solo un parámetro.
Esta condición sólo debe aplicarse si se desea generar un nuevo *array* a partir de una posición determinada, hasta el final del *array* en cuestión.

```
const personas = ["Greta", "Fer", "Nico", "July", "Laura"];

const otrasPersonas = personas.slice(2);

// El array 'otrasPersonas' contendrá: Nico, July, Laura
```



.splice()

El método `.splice()` (del inglés: *empalmar*) **permite remover uno o más elementos alojado(s) en cualquier posición del *array***.

Recibe dos parámetros:

1. En el primer parámetro, le indicamos el **índice del elemento a remover**.
2. En el segundo, **cuántos elementos quitaremos**.

En el ejemplo del siguiente slide, solo removemos un elemento del *array*: removemos a '`Fer`' del *array* `personas`.

Si volvemos a consultar el *array*, veremos que dicho elemento no figura más. Esto hace que `.splice()` sea definido como un **método destructivo, porque modifica el *array* original**.

Ejemplo:

```
const personas = ["Greta", "Fer", "Nico", "July", "Laura"];

const persona = personas.splice(1, 1);

// El array 'persona' contendrá: Fer

// En el array 'personas' quedarán: Greta, Nico, July, Laura
```

JS

El método `.splice()`, también es utilizado para **reemplazar elementos dentro de un *array*, por un nuevo elemento**.

En este otro ejemplo, reemplazamos el índice **2** del *array* **“Nico”**, por el elemento **“Nicolás”**.

Si queremos “capturar” el elemento reemplazado en el *array* `personas`, anteponemos a la cláusula **`personas.splice(...)`** una constante, donde quedará capturado el valor que cambiamos: **“Nico”**.

```
const personas = ["Greta", "Fer", "Nico", "July", "Laura"];

personas.splice(2, 1, "Nicolás");

// En el array 'personas' quedarán: Greta, Fer, Nicolás, July, Laura
```

.indexOf()

El método `indexOf()` permite **identificar si un elemento existe o no dentro de un *array***. Si existe, nos retorna su índice actual; si no, retornará **-1**.

Es una opción muy útil, que se puede aprovechar para **identificar el índice de un elemento que deseamos quitar**.

En el segundo ejemplo, retorna como resultado `-1`, porque el nombre a identificar no se encuentra dentro de los elementos del *array* en cuestión.

```
const personas = ["Greta", "Fer", "Nico", "July", "Laura"];

const idx = personas.indexOf("Laura");

// idx tendrá como valor: 4
```

```
const personas = ["Greta", "Fer", "Nico", "July", "Laura"];

const idx = personas.indexOf("Laica");

// idx tendrá como valor: -1
```

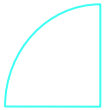
Ejemplo aplicado, combinando `.indexOf` y `.splice`:

```
const personas = ["Greta", "Fer", "Nico", "July", "Laura"];

let aQuitar = input("Ingresa el nombre de la persona a quitar:")

const resultado = personas.indexOf(aQuitar)

if (resultado === -1) {
    console.warn("La persona a quitar no encuentra cargada en el array.")
} else {
    personas.splice(aQuitar, 1)
}
```



.includes()

El método `.includes()` permite **identificar si un elemento existe dentro de un *array*, o no**.

A diferencia de `.indexOf()`, `.includes()` retorna solamente **`true`**, si existe, o **`false`**, si no existe.

```
const personas = ["Greta", "Fer", "Nico", "July", "Laura"];

const idx = personas.indexOf("Laica");

// idx tendrá como valor: -1
```

Ejemplo aplicado, combinando `.includes` y `.push`:

```
const personas = ["Greta", "Fer", "Nico", "July", "Laura"];

let nuevaPersona = input("Ingresa el nombre de la persona a quitar:")

const resultado = personas.includes(nuevaPersona)

if (resultado === true) {
    console.warn("La persona a agregar ya existe en el array.")
} else {
    personas.push(nuevaPersona)
}
```

.concat()

Este método permite **concatenar dos *arrays* en uno solo**. En el siguiente ejemplo, se concatenan el *array* llamado **`paisesAmericaDelSur`** con **`paisesAmericaDelNorte`** para conformar un nuevo *array* cuyo nombre es **`paisesAmerica`**.

De acuerdo al orden que deseamos para los elementos en el *array*, es como debemos utilizar los *arrays* a fusionar. El resultado del ejemplo, tendrá en los primeros lugares del nuevo *array*, los elementos de `paisesAmericaDelSur`.

```
const paisesAmericaDelSur = ["Argentina", "Bolivia", "Brasil", "Chile", "Colombia",
"Ecuador", "Guyana", "Paraguay", "Perú", "Surinam", "Uruguay", "Venezuela"];

const paisesAmericaDelNorte = ["Canadá", "Estados Unidos", "México", "Alaska"];

const paisesAmerica = paisesAmericaDelSur.concat(paisesAmericaDelNorte);
```

.join()

El método `.join()` **fusiona todos los elementos de un array en una cadena de texto (*string*)**.

Entre los paréntesis, podemos definir un carácter a usar para separar los elementos unos de otros.

```
const paises = ["Argentina", "Bolivia", "Brasil", "Chile", "Colombia", "Perú",
"Uruguay", "Venezuela"];

paises.join(" - ");
// 'Argentina - Bolivia - Brasil - Chile - Colombia - Perú - Uruguay - Venezuela'

paises.join(", ");
// 'Argentina, Bolivia, Brasil, Chile, Colombia, Perú, Uruguay, Venezuela'

paises.join(" | ");
// 'Argentina | Bolivia | Brasil | Chile | Colombia | Perú | Uruguay | Venezuela'
```

.sort()

El método `.sort()` **ordena** los elementos de un *array* de **forma ascendente**.

(índice)	Valor
0	'Greta'
1	'Fer'
2	'Nico'
3	'July'
4	'Laura'

```
personas.sort();
```

(índice)	Valor
0	'Fer'
1	'Greta'
2	'July'
3	'Laura'
4	'Nico'

.reverse()

El método `.reverse()` **invierte la posición** de los elementos de un *array*.

(índice)	Valor
0	'Greta'
1	'Fer'
2	'Nico'
3	'July'
4	'Laura'

```
personas.reverse();
```

(índice)	Valor
0	'Laura'
1	'July'
2	'Nico'
3	'Fer'
4	'Greta'

Encadenamiento de métodos

JS permite utilizar encadenamiento de métodos, tanto en las variables como también en los *arrays*. Así, se consiguen aplicar, **en una sola línea de código, transformaciones y acciones sobre los datos que se están manipulando**.

En *arrays* de elementos, se puede aprovechar, como vemos en el ejemplo de la derecha, a combinar el método `sort()` junto a `reverse()` para conseguir un ordenamiento descendente de los elementos del *array*.

```
personas.sort().reverse();
```

(índice)	Valor
0	'Nico'
1	'Laura'
2	'July'
3	'Greta'
4	'Fer'

Este es otro ejemplo de encadenamiento de métodos sobre una **constante**. Aquí, aplicamos la eliminación de espacios (`trim`) y la transformación del texto a minúsculas (`toLowerCase`), para recién luego utilizar el valor de la constante y verificar si en un *array* existe o no elemento con dicho valor. **El orden de ejecución de los métodos encadenados, es el mismo que en el orden que los escribimos**.

```
const condimentos = ["pimienta", "perejíl", "albahaca"];

let aBuscar = "   ALBAHACA     ";

condimentos.includes(aBuscar);  //retornará false

condimentos.includes(aBuscar.trim().toLowerCase());  //retornará true
```

Atención:

Casi todos estos métodos de *arrays*, son totalmente **aplicables tanto en *arrays* de elementos** como también en ***array* de objetos**.

En el caso de *array* de objetos, existen **métodos alternativos para realizar determinadas operaciones** sobre los datos, que reemplazan a algunos métodos de los *arrays* de elementos, dado que un *array* de objetos tiene otro nivel de complejidad.

Revisión

- Repasar el concepto de ***arreglo* en JavaScript**.
- Mostrar una **lista de empleados**.
- Practicar la combinación de los **diferentes métodos de *arrays***.
- Trabajar en el ***Proyecto integrador***.
- Realizar las preguntas necesarias a la/el docente antes de continuar.

¡Sigamos trabajando!