Arrays

DEV.F
DESARROLLAMOS(PERSONAS);

Arreglos o Matrices (Arrays)

Los Arreglos “**son una manera ordenada”** de almacenar una lista de elementos de datos bajo un solo nombre de variable, pudiendo acceder a cada elemento individual de la lista.

Creación de un Arreglo

Un arreglo se representa con corchetes [ ], dentro se coloca el contenido. Cada elemento es separado por coma.

```
const alumnosMali = []; // arreglo vacio

alumnosMali = ['Willinton','Laura','Jorge','Luis','Rick'];
```

Los elementos incluso pueden ser de diferente tipo:

```
const cajaRevuelta = ['Palabra', [1,2], 1, 1.2, true, alumnosMali, null, { 'clave': 2 } ];
```

Acceder a los valores de un Arreglo

Podemos acceder a cada contenido individual indicando la posición numérica del elemento que queremos acceder entre corchetes [ ] (esto se llama **índice** o **index**). Importante: La primera posición es 0.

```
> var frutas = ['Pera','Manzana','Platano','Naranja'];
> frutas
<· ▶(4) ["Pera", "Manzana", "Platano", "Naranja"]
```

Posición	0	1	2	3

```
> frutas[2]
<· "Platano"
```

Modificar un valor de un Arreglo

Podemos modificar el valor de un elemento individual asignando un nuevo valor a una posición determinada del arreglo, indicada entre corchetes [ ].

```
> var frutas = ['Pera','Manzana','Platano','Naranja'];
```

Posición	0	1	2	3

```
> frutas[3] = "Uvas";
<· "Uvas"

> frutas
<· ▶ (4) ["Pera", "Manzana", "Platano", "Uvas"]
```

length

La propiedad **length** nos devuelve el número total de elementos en el arreglo.

Este método es indispensable para poder iterar (recorrer) el arreglo y hacer operaciones con dichos elementos (se verá más adelante).

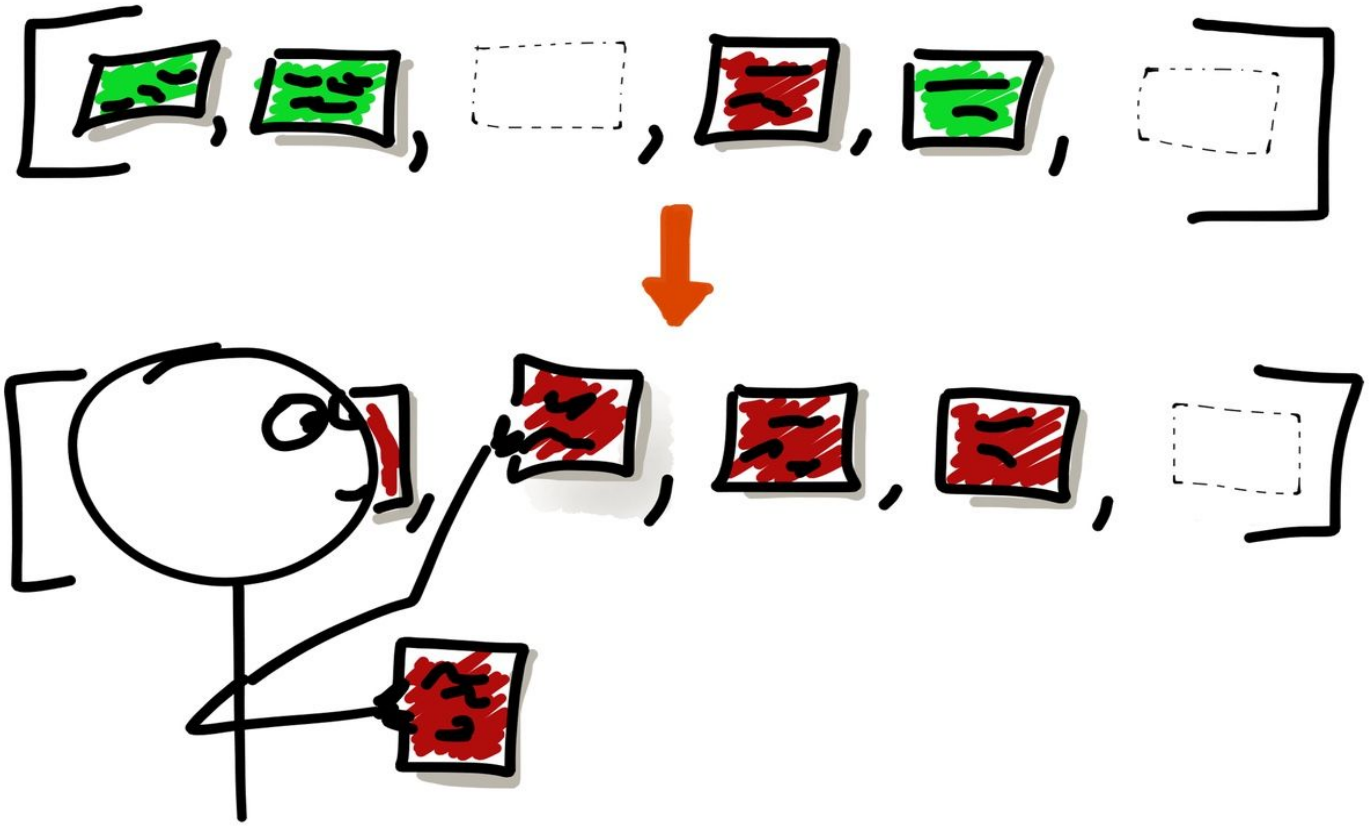
```
> var frutas = ['Pera','Manzana','Platano','Uvas'];

    > frutas.length;
    < 4
```

Métodos de Arrays

DEV.F:
DESARROLLAMOS(PERSONAS);

Métodos de Arreglos (Arrays)



push y pop

```
> frutas;
<· ▶ (4) ["Pera", "Manzana", "Platano", "Uvas"]
```

El método **push agrega** un ítem **al final de la lista**.

```
> frutas.push("Mandarina");
<· 5
> frutas;
<· ▶ (5) ["Pera", "Manzana", "Platano", "Uvas", "Mandarina"]
```

El método **pop elimina** el ítem que está a**l final de la lista.**

```
> frutas.pop();
<· "Mandarina"
> frutas;
<· ▶ (4) ["Pera", "Manzana", "Platano", "Uvas"]
```

unshift y shift

```
> frutas;
<· ▶ (4) ["Pera", "Manzana", "Platano", "Uvas"]
```

El método **unshift agrega** un ítem **al principio de la lista**.

```
> frutas.unshift("Mandarina");
<· 5
> frutas;
<· ▶ (5) ["Mandarina", "Pera", "Manzana", "Platano", "Uvas"]
```

El método **shift elimina** el ítem que está **al principio de la lista.**

```
> frutas.shift();
<· "Mandarina"
> frutas;
<· ▶ (4) ["Pera", "Manzana", "Platano", "Uvas"]
```

split

Divide una cadena (string) en una matriz de subcadenas, tomando como referencia donde encuentre un carácter indicado.

```
> var verduras = "Cebolla,Perejil,Tomate,Calabaza";

> var arregloVerduras = verduras.split(',');

> arregloVerduras;
<- ▶ (4) ["Cebolla", "Perejil", "Tomate", "Calabaza"]
```

Slice (porcion)

Quita una parte de una cadena y **devuelve una nueva cadena.**

```
> var verduras = ['Cebolla','Perejil','Tomate','Calabaza','Brocoli'];
```

Debe indicarse al menos una posición inicial (start). La posición inicial es 0.

```
> verduras.slice(2);
◂· ▶ (3) ["Tomate", "Calabaza", "Brocoli"]
```

Opcionalmente también se puede indicar una posición final (end).

```
> verduras.slice(1,3);
◂· ▶ (2) ["Perejil", "Tomate"]
```

splice

Sirve para agregar o borrar elementos de un arreglo. Pide como parámetros el **index** y un **número** de elementos a borrar. Splice modifica el arreglo original.

```
> var verduras = ['Cebolla','Perejil','Tomate','Calabaza','Brocoli'];
> verduras.splice(2,0,"Pepino","Limon");
<· ▶ []
> verduras;
<· ▶ (7) ["Cebolla", "Perejil", "Pepino", "Limon", "Tomate", "Calabaza", "Brocoli"]
```

Retorna los elementos borrados (si hubiese).

```
> verduras.splice(2,1,"Pepino","Limon");
<· ▶ ["Tomate"]
verduras;
▶ (6) ["Cebolla", "Perejil", "Pepino", "Limon", "Calabaza", "Brocoli"]
```

sort

Ordena la lista de forma ascendente (A-Z) por defecto.

```
> var verduras = ['Cebolla','Perejil','Tomate','Calabaza','Brocoli'];

> verduras.sort();
◂ ▶ (5) ["Brocoli", "Calabaza", "Cebolla", "Perejil", "Tomate"]
```

Es posible pasarle una función para ajustar el orden. Sobre todo para números, ya que por defecto no los ordena correctamente.

```
> [2,5,1,3,46,70,34].sort();
◂ ▶ (7) [1, 2, 3, 34, 46, 5, 70]

> [2,5,1,3,46,70,34].sort(function(a, b){return a-b});
◂ ▶ (7) [1, 2, 3, 5, 34, 46, 70]
```

reverse

Coloca los elementos del arreglo al revés. Este método altera el arreglo original.

```
> var verduras = ['Cebolla','Perejil','Tomate','Calabaza','Brocoli'];
> verduras.reverse();
<· ▶ (5) ["Brocoli", "Calabaza", "Tomate", "Perejil", "Cebolla"]
```

```
> [1,2,3,4,7,8,9].reverse()
<· ▶ (7) [9, 8, 7, 4, 3, 2, 1]
```

concat

Este método une (concatena) el contenido de 2 arreglos existentes. **No modifica dichos arreglos**, si no que devuelve uno nuevo.

```
> var verduras = ['Cebolla','Perejil','Tomate'];
> var frutas = ['Manzana','Pera','Platano'];

> var listaDeCompras = verduras.concat(frutas);

> listaDeCompras;
< ▶ (6) ["Cebolla", "Perejil", "Tomate", "Manzana", "Pera", "Platano"]
```