BUCLES Y ARRAYS

Lo que no vimos en clase...

TABLA DE CONTENIDOS

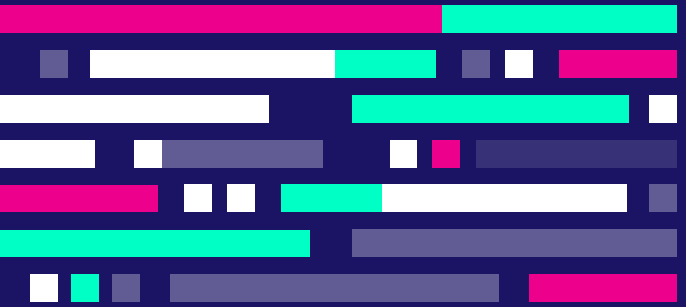
INTRODUCCIÓN

Esto es lo que deberían hacer en Programación Imperativa :)

```
if(person.sad) {
    person.stopSadness();
    person.beAwesome();
}
```

01

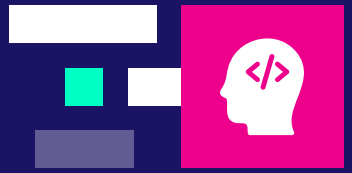
BUCLES



¿PARA QUÉ NOS SIRVEN?

Para repetir una operación las veces que querramos

CÓMO DECLARAR BUCLES

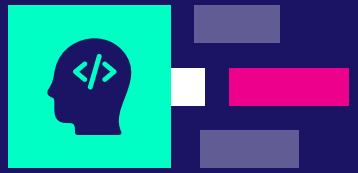


For

```
for ([expresiónInicial]; [expresiónCondicional]; [expresiónDeActualización]) {
      // instruccion
}
```

While

```
while (condicion) {
  // instruccion
}
```



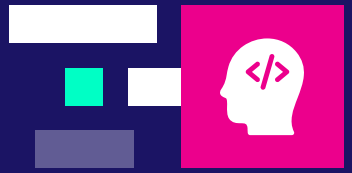
Do While

```
do {
  // instruccion
} while (condicion);
```

02

MÉTODOS DE ARRAYS I

AGREGAR Y SACAR ELEMENTOS

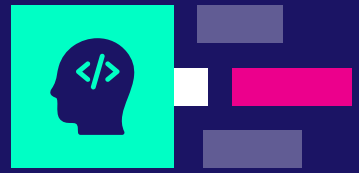


push(...items)

Agrega elementos al final del array

pop()

Saca el último elemento del array

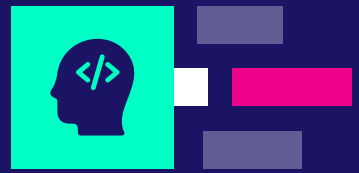


shift(...items)

Agrega elementos al principio del array

unshift()

Saca el primer elemento del array



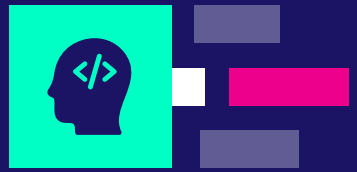
MÉTODOS BÁSICOS PARA ARRAYS

slice(principio, final)

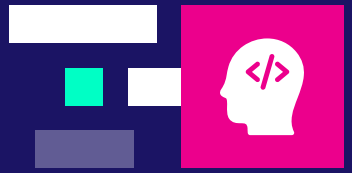
Devuelve un **nuevo array** copiando todos los elementos desde principio hasta final (sin incluir a final)

concat(...arrays)

Devuelve un **nuevo array** que incluye los valores del array que hizo la llamada Y los arrays recibidos



BUSCANDO

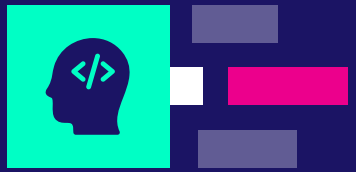


indexOf(elemento, desde)

Busca **elemento** comenzando en el índice **desde** (por defecto es 0). Si lo encuentra devuelve el index, sino -1

lastIndexOf(elemento, desde)

Busca **elemento** comenzando en el índice **desde** (por defecto es 0). Si lo encuentra devuelve el **último** index, sino -1



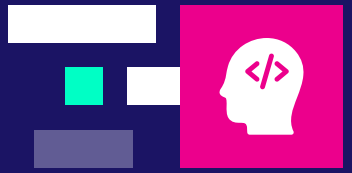
includes(elemento, desde)

Busca **elemento** comenzando en el índice **desde** (por defecto es 0). Si lo encuentra devuelve true, sino false

03

MÉTODOS DE ARRAYS II

FOR EACH



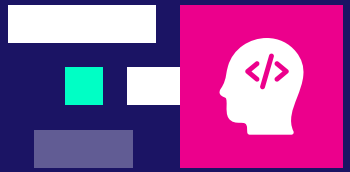
forEach(cb)

No devuelve nada. Simplemente itera sobre un array

```
function iterar(array, cb) {
  for (let i = 0; i < array.length; i++) {
    cb(array[i]));
  }
}
```

```
array.forEach(function (elemento) {
  // lo que queremos que se haga con cada elemento
});
array.forEach(elemento => {
  // lo que queremos que se haga con cada elemento
});
```

FILTER



filter(cb)

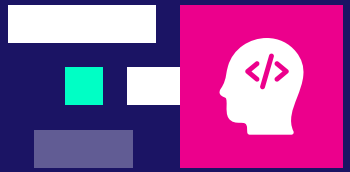
Devuelve un **nuevo array** que contiene únicamente los elementos que cumplan con la condición del callback

```
function filtrar(array, cb) {
  let arrayFiltrado = [];
  for (let i = 0; i < array.length; i++) {
    if (cb(array[i])) {
      arrayFiltrado.push(array[i]);
    }
  }
  return arrayFiltrado;
}
```

```
array.filter(function (elemento) {
  // definimos la condición de filtro
});

array.filter(elemento => {
  // definimos la condición de filtro
});
```

MAP



map(cb)

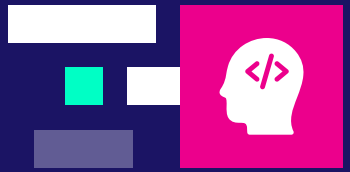
Devuelve un **nuevo array** que contiene los elementos del original **modificados**

```
function mapear(array, cb) {
  let arrayMapeado = [];
  for (let i = 0; i < array.length; i++) {
    let elemento = array[i];
    let elementoTransformado = cb(elemento);
    arrayMapeado .push(elementoTransformado);
  }
  return arrayMapeado ;
}
```

```
array.map(function (elemento) {
  // las modificaciones sobre elemento
});

array.map(elemento => {
  // las modificaciones sobre elemento
});
```

FIND



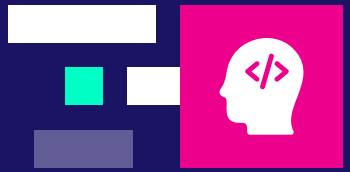
find(cb)

Devuelve **el valor** del primer elemento que cumpla con el callback

```
function encontrar(array, cb) {
  let valor = null;
  for (let i = 0; i < array.length; i++) {
    if (cb(array[i])) {
      valor = array[i];
    }
  }
  return valor;
}
```

```
array.find(function (elemento) {
  // condición para encontrar el elemento
});
array.find(elemento => {
  // condición para encontrar el elemento
});
```

FINDINDEX



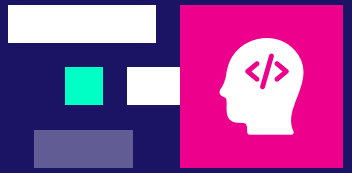
findIndex(cb)

Devuelve **el índice** del primer elemento que cumpla con el callback

```
function encontrarIndice(array, cb) {
  let indice = -1;
  for (let i = 0; i < array.length; i++) {
    if (cb(array[i])) {
      indice = i;
    }
  }
  return indice;
}
```

```
array.findIndex(function (elemento) {
  // condición para encontrar el elemento
});
array.findIndex(elemento => {
  // condición para encontrar el elemento
});
```

REDUCE



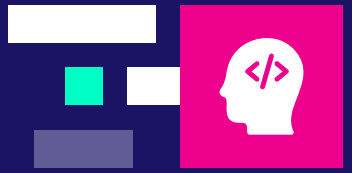
reduce(cb)

Devuelve un **valor del array reducido**. Éste método va a recorrer el array de izquierda a derecha pasando por todos sus elementos acumulativamente

```
function reducir(array, cb) {
  let acumulador = null;
  for (let i = 0; i < array.length; i++) {
    cb(acumulador, array[i]);
  }
  return acumulador;
}
```

```
array.reduce(function(acum, elemento) {
  // definimos como queremos que se
acumulen los elementos
});
array.reduce((acum, elemento) => {
  // definimos como queremos que se
acumulen los elementos
});
```

SORT



sort(cb)

No devuelve nada. Éste método va a ordenar los elementos del array utilizando como criterio de comparación el callback que recibe. Por defecto responde a la posición del valor del string de acuerdo a su valor Unicode

```
function ordenar(array, cb) {
  for (let i = 0; i < array.length; i++) {
    for (let j = 0; j < array.length; j++) {
      array[i] = cb(array[i], array[j]);
    }
  }
}
```

```
array.sort(function(elem1, elem2) {
  // definimos como queremos que se realice la comparación entre elem1 y elem2
});
array.sort((elem1, elem2) => {
 // definimos como queremos que se realice la comparación entre elem1 y elem2
});
```

IMPORTANTE

El callback que definamos **va a tener que devolver una de las siguientes opciones**:

- Un número **negativo** (ej: -1) si **elem1 < elem2** (osea: si queremos que elem1 se ordene antes que elem2)
- Un número **positivo** (ej: 1) si **elem1 > elem2** (osea: si queremos que elem2 se ordene antes que elem1)

Si devolvemos 0 no cambia nada

```
array.sort((elem1, elem2) => {
    let resultado = 0;
    if (elem1.propiedad.length < elem2.propiedad.length)
        resultado = 1;
    else
        resultado = -1;
    return resultado;
});
```

¿QUÉ ESTÁ PASANDO AHÍ?

Inicializamos resultado en 0

Si la longitud de propiedad del **elem1** es **MENOR** que la del **elem2** → **resultado = 1**
Porque quiero que el orden sea: elem2, elem1

Sino quiere decir que la longitud de propiedad del **elem1** es **MAYOR** que la del **elem2** → **resultado = -1**
Porque quiero que el orden sea: elem2, elem1

Supongamos que tenemos el siguiente array:

```
[
    {
        nombre: 'Pepa',
        notas: [ 7, 4, 5, 6, 7, 8 ]
    },
    {
        nombre: 'Pepo',
        notas: [ 10, 8 ]
    },
    {
        nombre: 'Pipo',
        notas: [ 10, 10, 10, 10 ]
    }
]
```

Y queremos ordenarlo con el callback anterior

```
array.sort((alumno1, alumno2) => {
    let resultado = 0;
    if (alumno1.notas.length < alumno2.notas.length)
        resultado = 1;
    else
        resultado = -1;
    return resultado;
});
```

¿CÓMO QUEDARÍA?

El array se ordena **de mayor a menor por longitud de array**

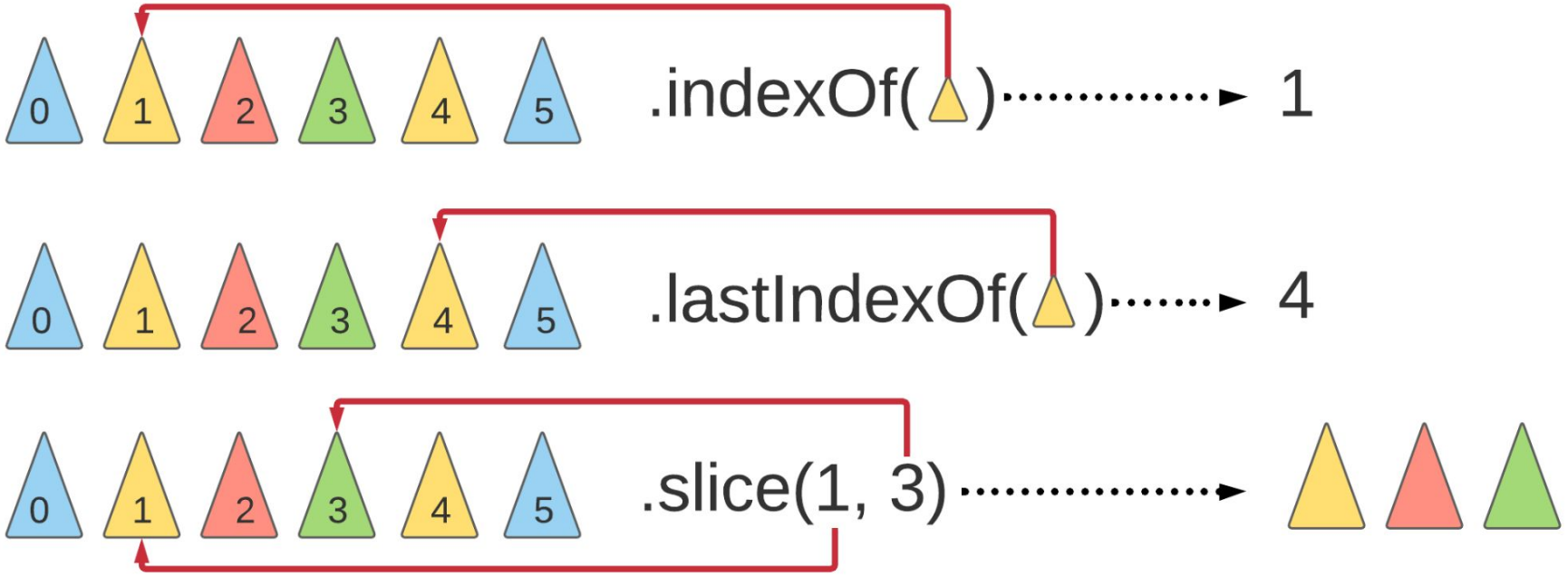
```
[
    {
        nombre: 'Pepa',
        notas: [ 7, 4, 5, 6, 7, 8 ]
    },
    {
        nombre: 'Pepo',
        notas: [ 10, 8 ]
    },
    {
        nombre: 'Pipo',
        notas: [ 10, 10, 10, 10 ]
    }
]
```

```
[
    {
        nombre: 'Pepa',
        notas: [ 7, 4, 5, 6, 7, 8 ]
    },
    {
        nombre: 'Pipo',
        notas: [ 10, 10, 10, 10 ]
    },
    {
        nombre: 'Pepo',
        notas: [ 10, 8 ]
    }
]
```

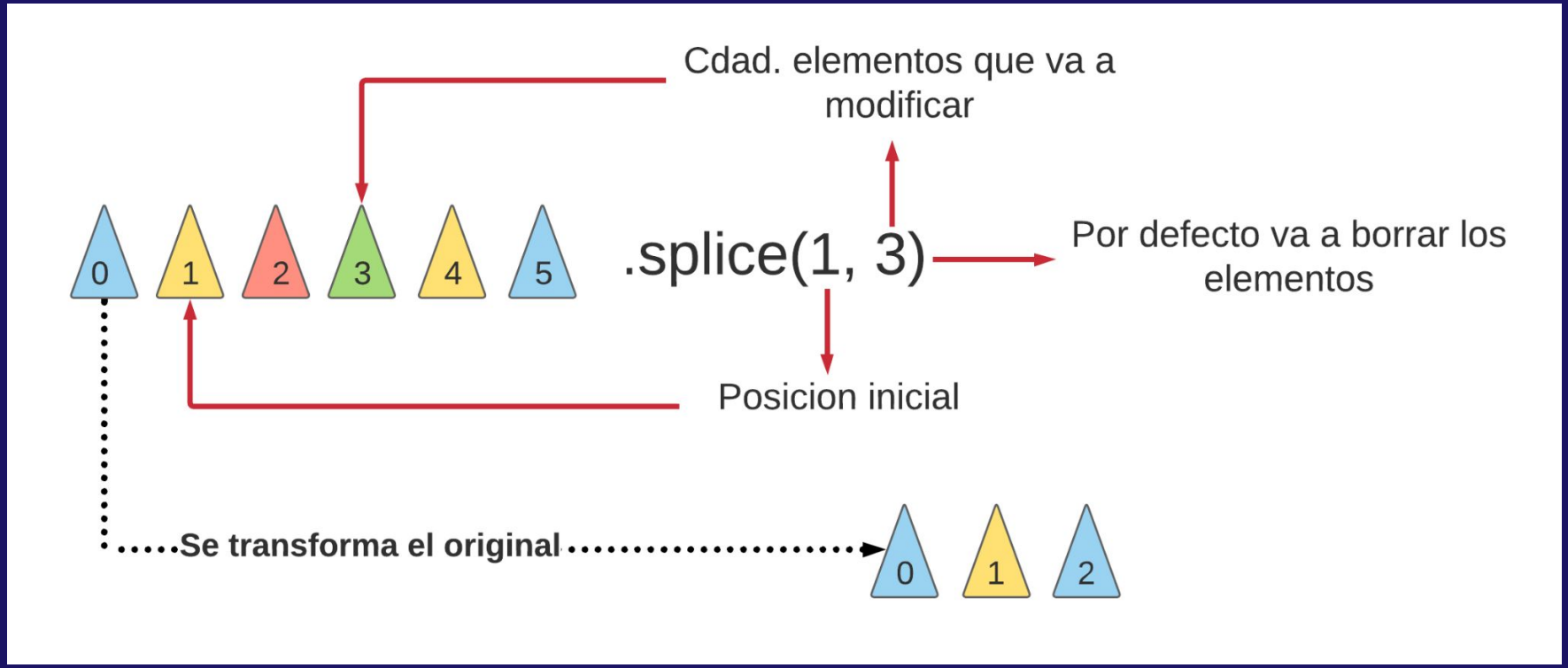
04

RESUMEN

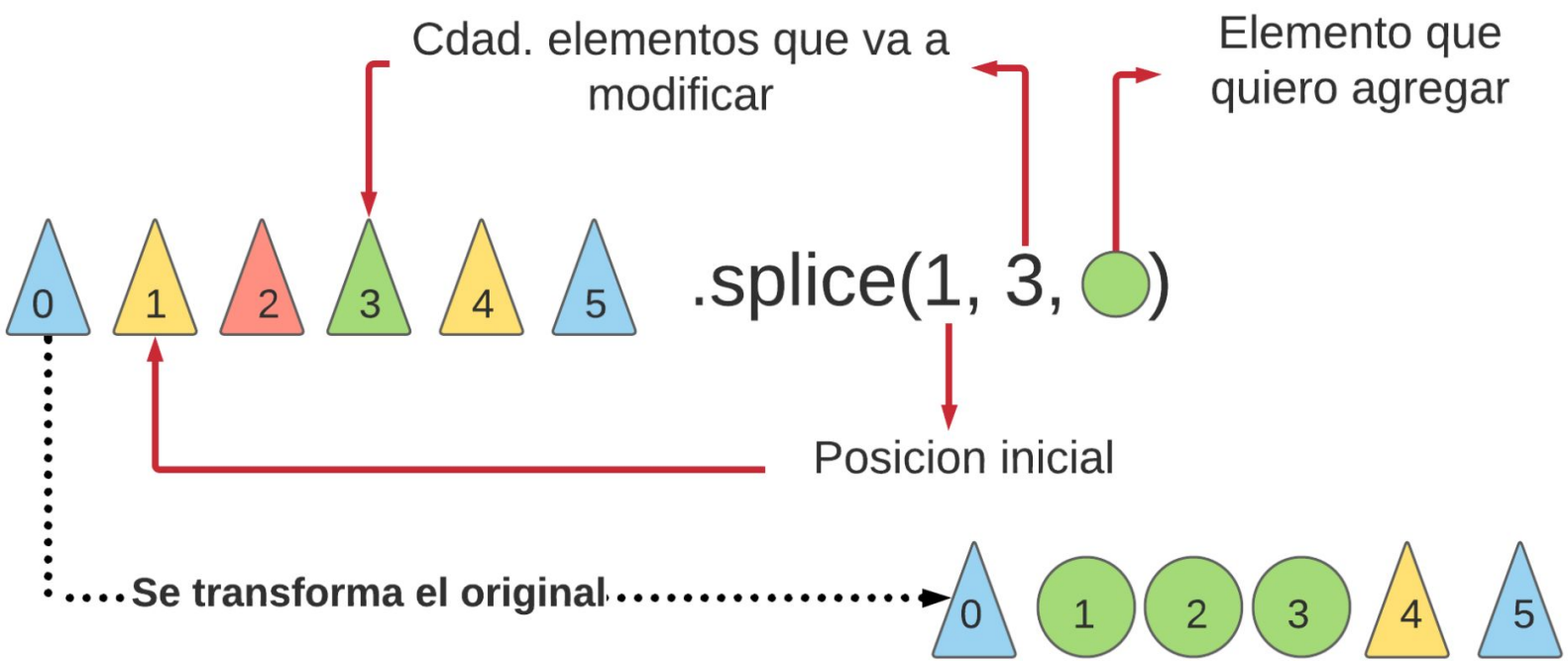
MÉTODOS QUE RECIBEN VALORES



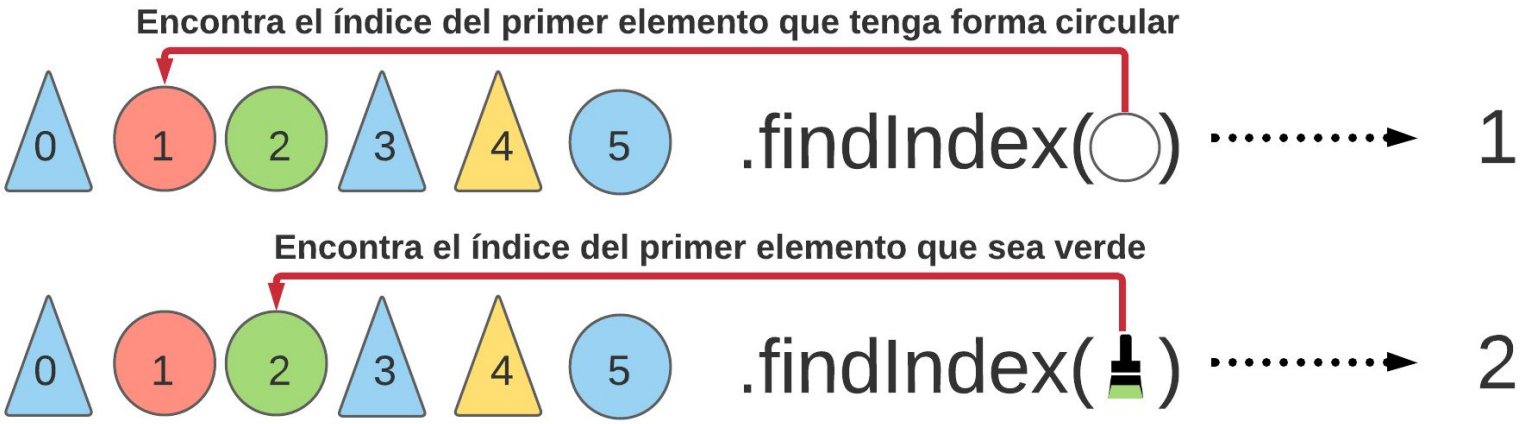
MÉTODOS QUE RECIBEN VALORES



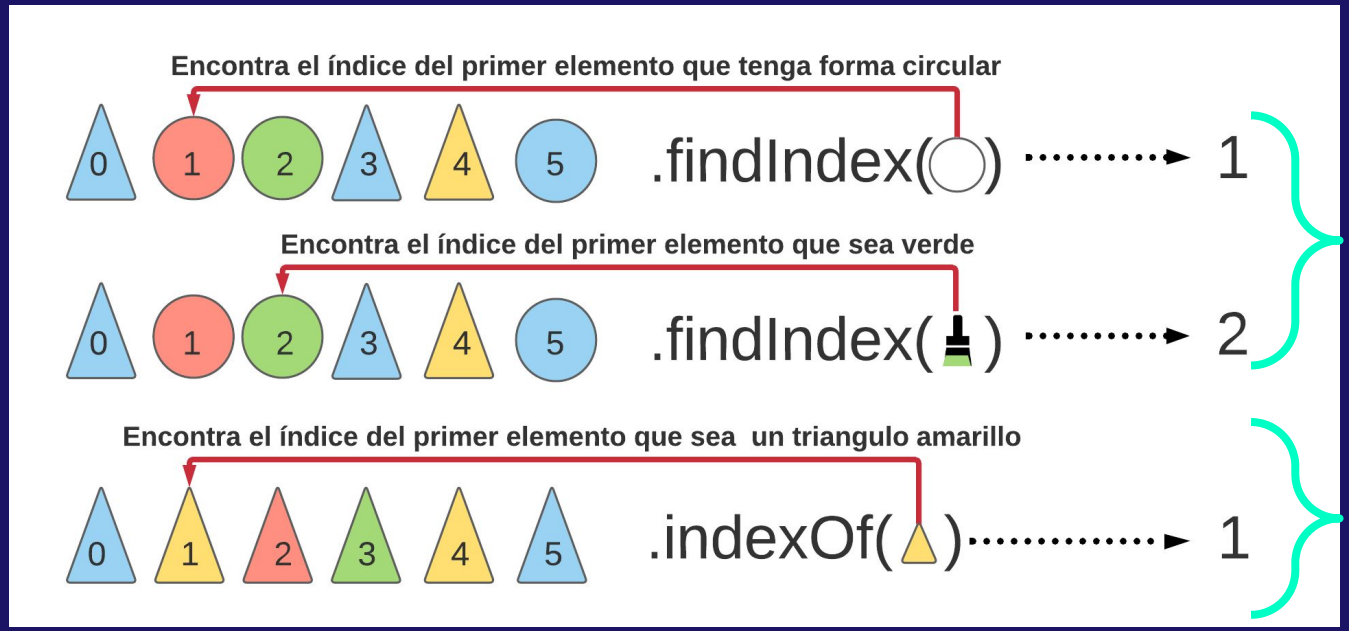
MÉTODOS QUE RECIBEN VALORES



MÉTODOS QUE RECIBEN CALLBACKS



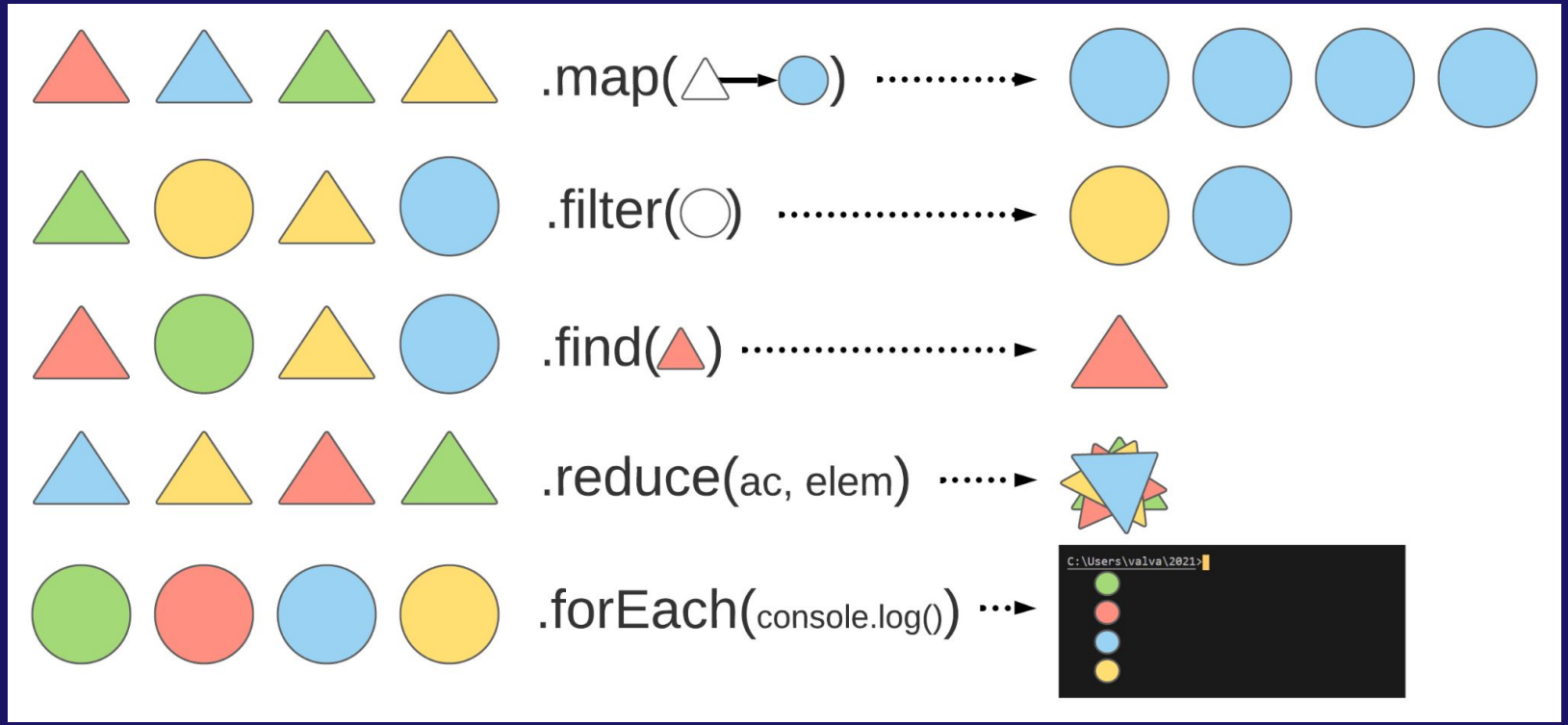
indexOf() vs findIndex()

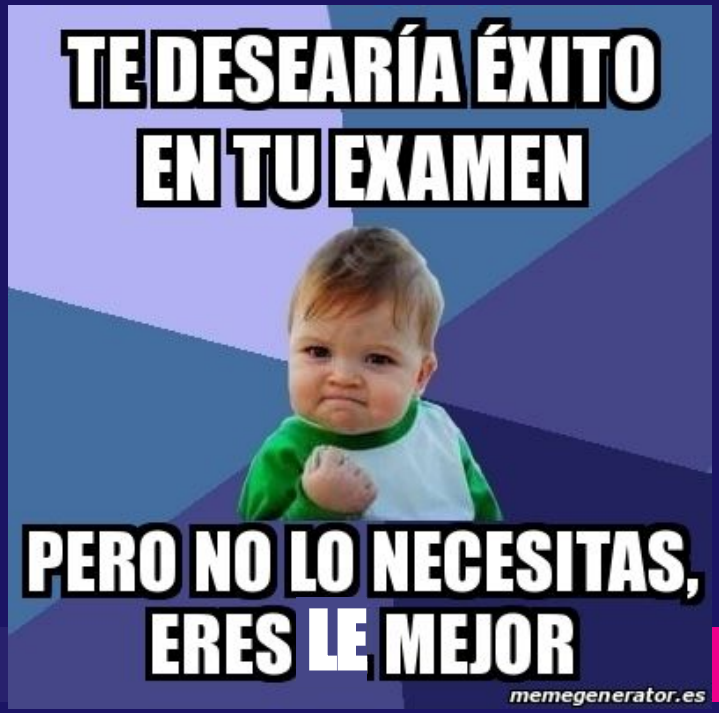


Recibe un callback, lo que nos permite refinar la búsqueda de la forma que queramos

Recibe un **valor,** va a buscar el elemento que sea exactamente igual al recibido

MÉTODOS QUE RECIBEN CALLBACKS




TE DESEARÍA ÉXITO
EN TU EXAMEN
PERO NO LO NECESITAS,
ERES LE MEJOR
memegenerator.es