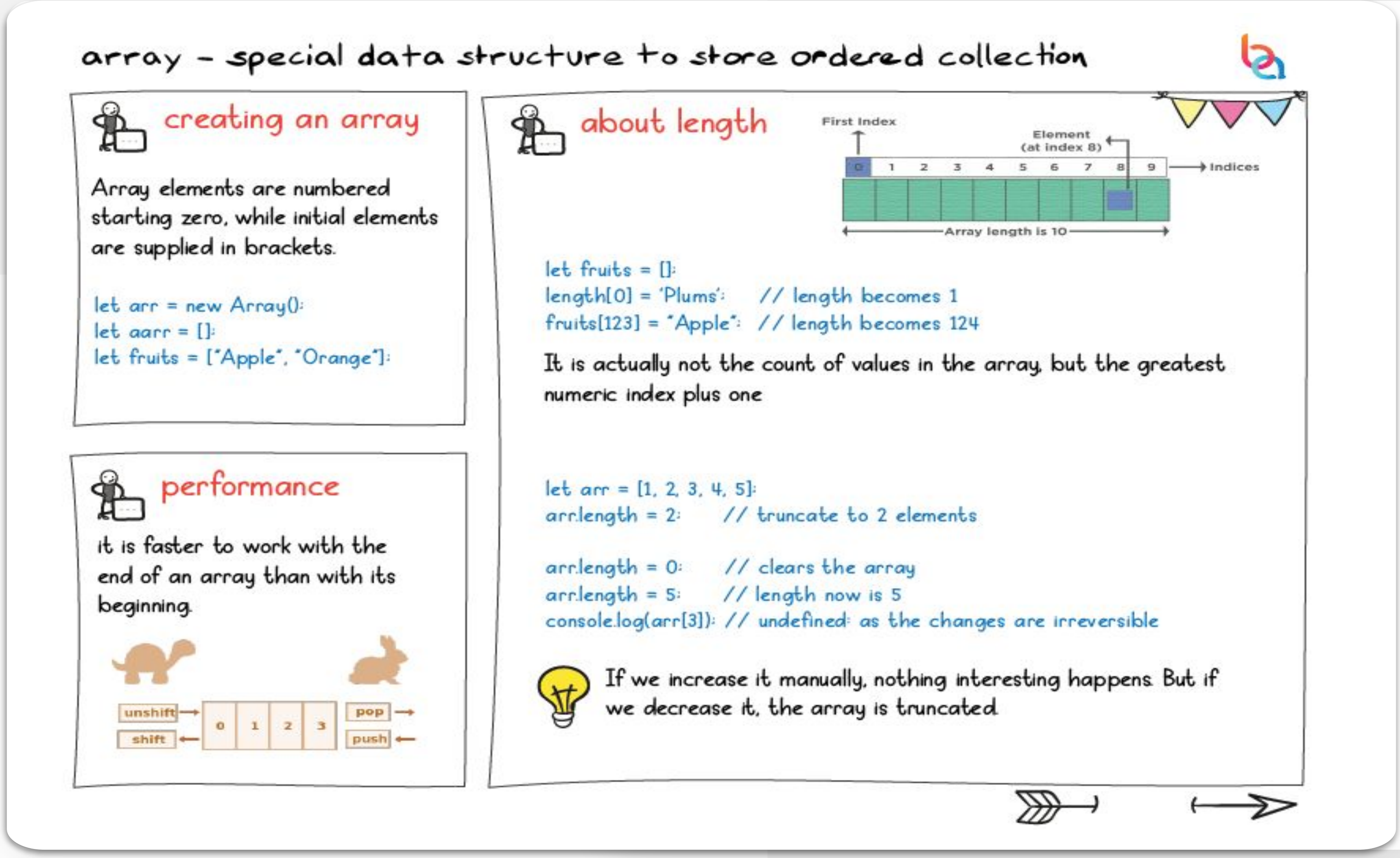
BAssure

NGS

JAVASCRIPT : ARRAY



array - special data structure to store ordered collection

creating an array

Array elements are numbered starting zero, while initial elements are supplied in brackets.

```
let arr = new Array();
let aarr = [];
let fruits = ["Apple", "Orange"];
```

performance

it is faster to work with the end of an array than with its beginning.

unshift
shift
0
1
2
3
pop
push

about length

First Index
Element
(at index 8)
0
1
2
3
4
5
6
7
8
9
Indices
Array length is 10

```
let fruits = [];
length[0] = 'Plums';     // length becomes 1
fruits[123] = "Apple";  // length becomes 124
```

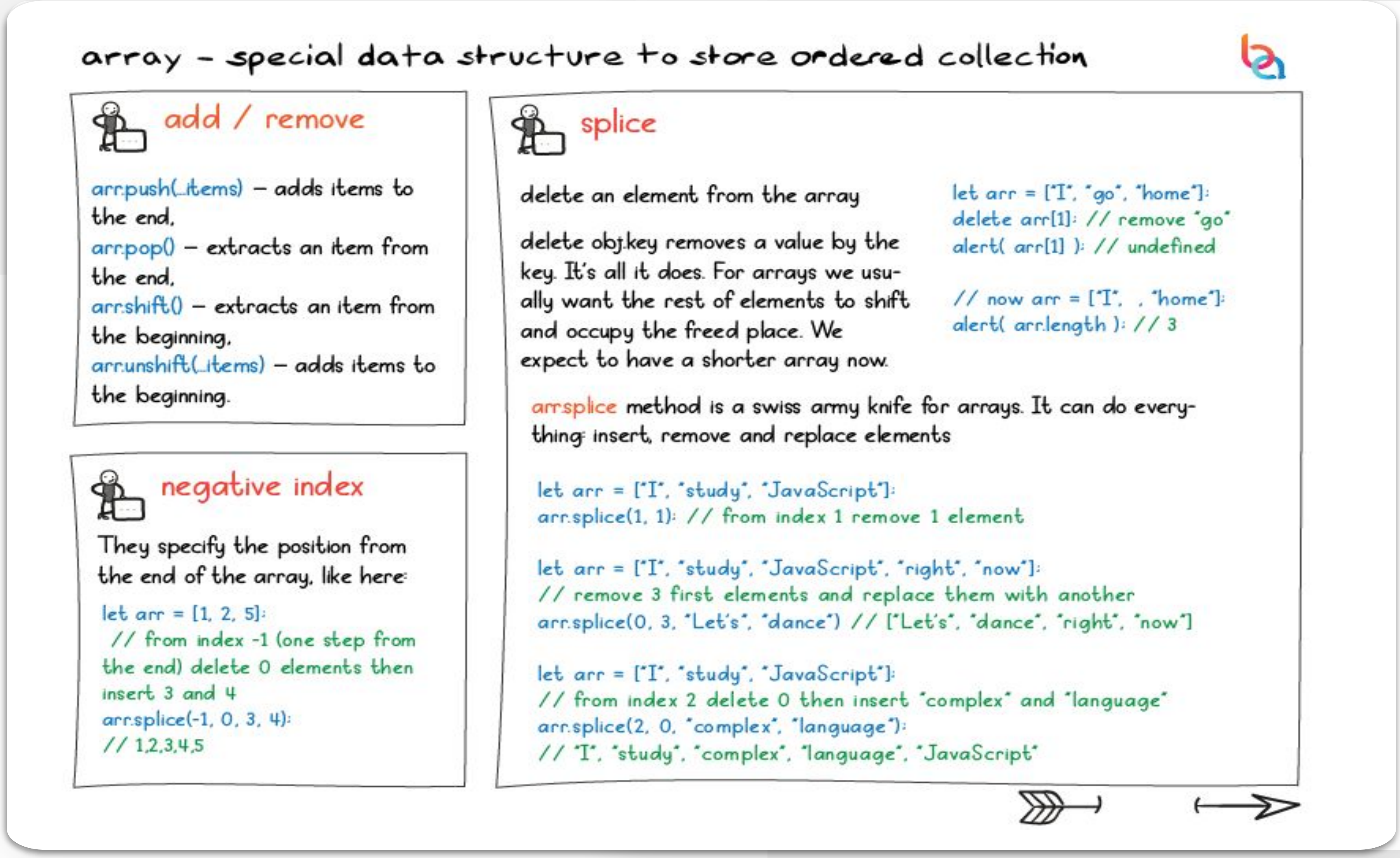
It is actually not the count of values in the array, but the greatest numeric index plus one

```
let arr = [1, 2, 3, 4, 5];
arr.length = 2;      // truncate to 2 elements

arr.length = 0;      // clears the array
arr.length = 5;      // length now is 5
console.log(arr[3]); // undefined: as the changes are irreversible
```

If we increase it manually, nothing interesting happens. But if we decrease it, the array is truncated.

array - special data structure to store ordered collection



array - special data structure to store ordered collection

add / remove

arr.push(...items) – adds items to the end,
arr.pop() – extracts an item from the end,
arr.shift() – extracts an item from the beginning,
arr.unshift(...items) – adds items to the beginning.

negative index

They specify the position from the end of the array, like here:

```
let arr = [1, 2, 5];
 // from index -1 (one step from
the end) delete 0 elements then
insert 3 and 4
arr.splice(-1, 0, 3, 4);
// 1,2,3,4,5
```

splice

delete an element from the array

delete obj.key removes a value by the key. It's all it does. For arrays we usually want the rest of elements to shift and occupy the freed place. We expect to have a shorter array now.

```
let arr = ["I", "go", "home"];
delete arr[1]; // remove "go"
alert( arr[1] ); // undefined

// now arr = ["I",  , "home"];
alert( arr.length ); // 3
```

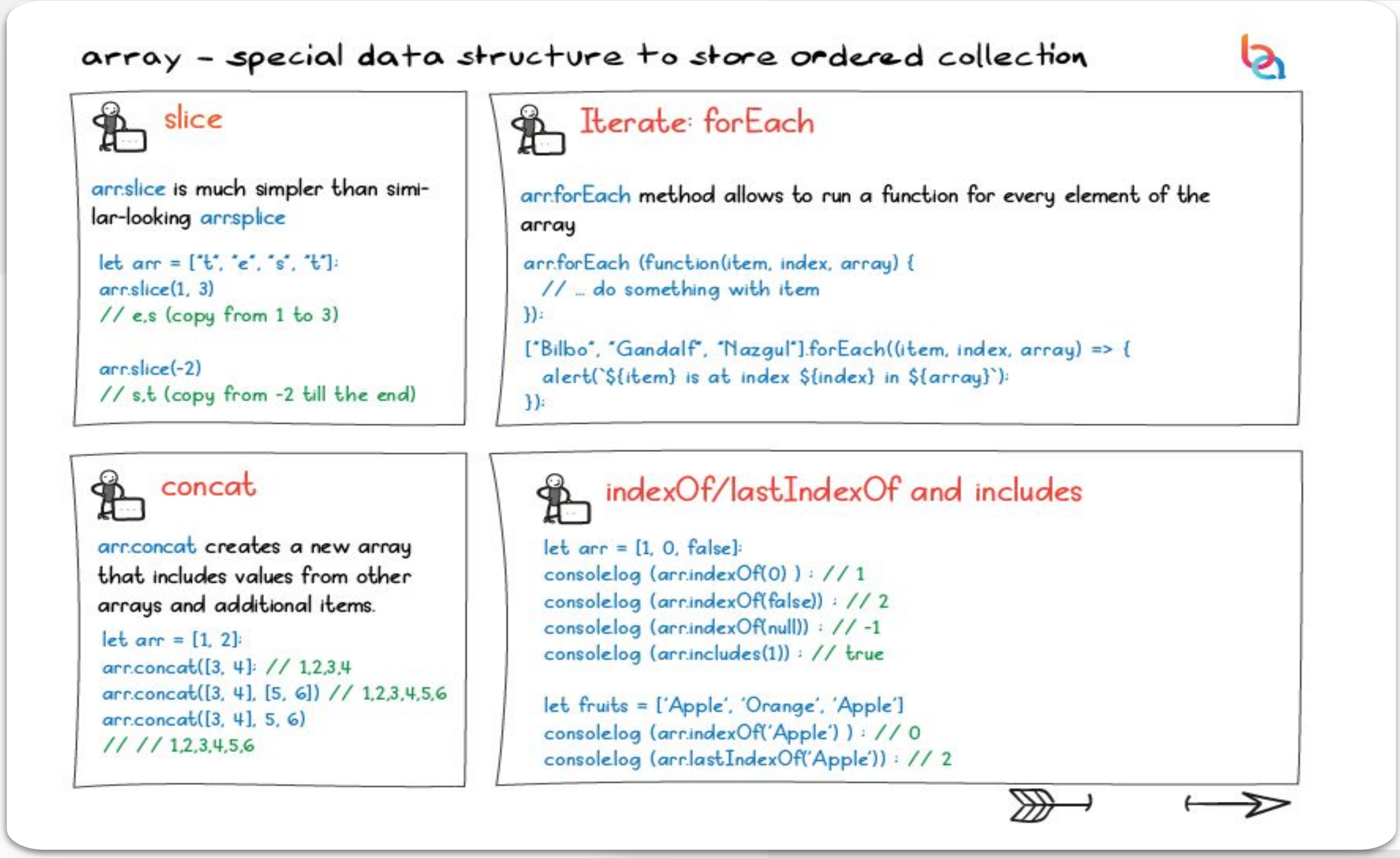
arr.splice method is a swiss army knife for arrays. It can do everything: insert, remove and replace elements

```
let arr = ["I", "study", "JavaScript"];
arr.splice(1, 1); // from index 1 remove 1 element

let arr = ["I", "study", "JavaScript", "right", "now"];
// remove 3 first elements and replace them with another
arr.splice(0, 3, "Let's", "dance") // ["Let's", "dance", "right", "now"]

let arr = ["I", "study", "JavaScript"];
// from index 2 delete 0 then insert "complex" and "language"
arr.splice(2, 0, "complex", "language");
// "I", "study", "complex", "language", "JavaScript"
```

array - special data structure to store ordered collection



array - special data structure to store ordered collection

slice

arr.slice is much simpler than similar-looking arr.splice

```
let arr = ["t", "e", "s", "t"];
arr.slice(1, 3)
// e,s (copy from 1 to 3)

arr.slice(-2)
// s,t (copy from -2 till the end)
```

Iterate: forEach

arr.forEach method allows to run a function for every element of the array

```
arr.forEach (function(item, index, array) {
  // ... do something with item
});
["Bilbo", "Gandalf", "Nazgul"].forEach((item, index, array) => {
  alert(`${item} is at index ${index} in ${array}`);
});
```

concat

arr.concat creates a new array that includes values from other arrays and additional items.

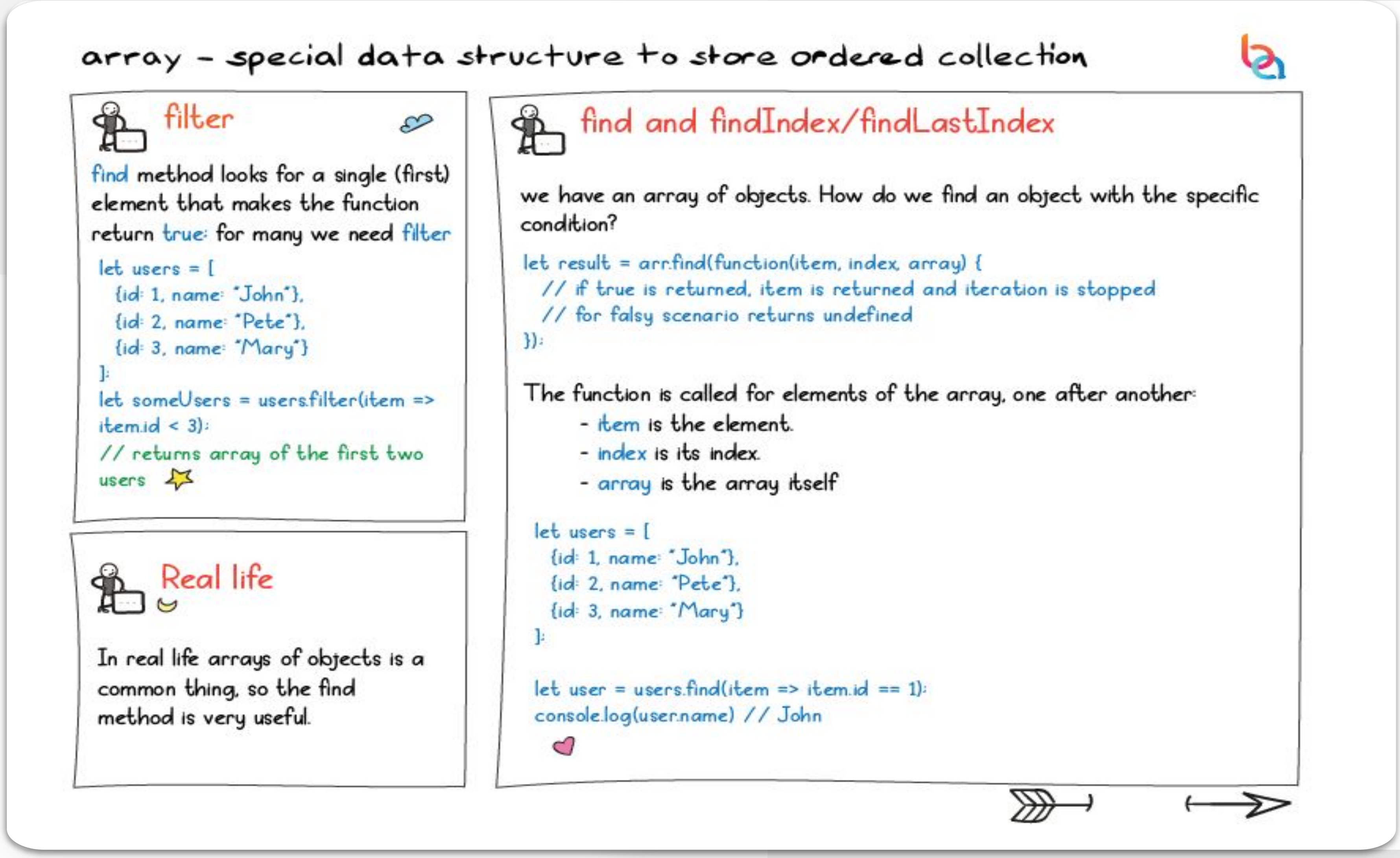
```
let arr = [1, 2];
arr.concat([3, 4]; // 1,2,3,4
arr.concat([3, 4], [5, 6]) // 1,2,3,4,5,6
arr.concat([3, 4], 5, 6)
// // 1,2,3,4,5,6
```

indexOf/lastIndexOf and includes

```
let arr = [1, 0, false];
console.log (arr.indexOf(0) ) ; // 1
console.log (arr.indexOf(false)) ; // 2
console.log (arr.indexOf(null)) ; // -1
console.log (arr.includes(1)) ; // true

let fruits = ['Apple', 'Orange', 'Apple']
console.log (arr.indexOf('Apple') ) ; // 0
console.log (arr.lastIndexOf('Apple')) ; // 2
```

array - special data structure to store ordered collection



array - special data structure to store ordered collection

filter

find method looks for a single (first) element that makes the function return true; for many we need filter

```
let users = [
  {id: 1, name: "John"},
  {id: 2, name: "Pete"},
  {id: 3, name: "Mary"}
];
let someUsers = users.filter(item =>
item.id < 3);
// returns array of the first two
users
```

Real life

In real life arrays of objects is a common thing, so the find method is very useful.

find and findIndex/findLastIndex

we have an array of objects. How do we find an object with the specific condition?

```
let result = arr.find(function(item, index, array) {
  // if true is returned, item is returned and iteration is stopped
  // for falsy scenario returns undefined
});
```

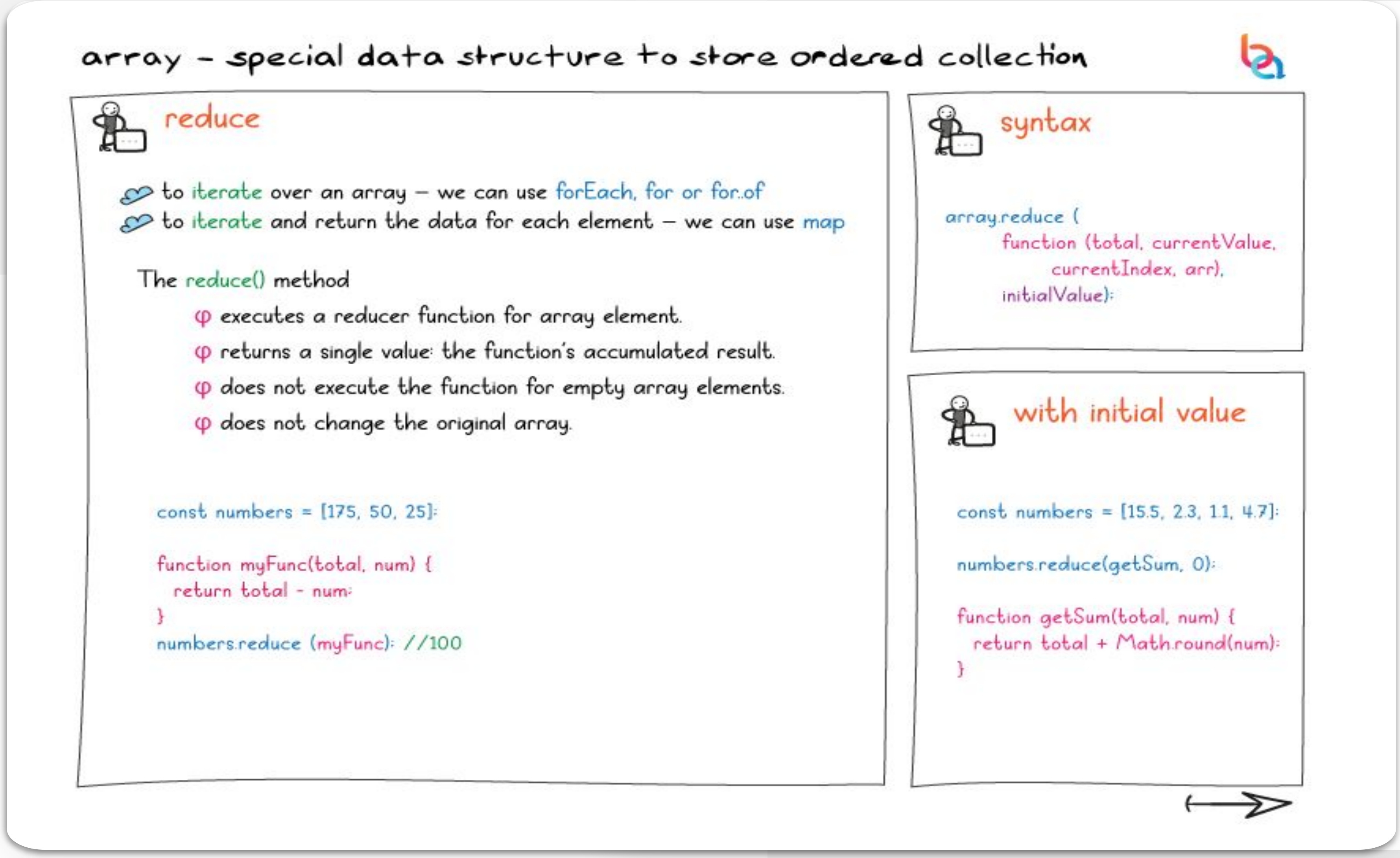
The function is called for elements of the array, one after another:

- item is the element.
- index is its index.
- array is the array itself

```
let users = [
  {id: 1, name: "John"},
  {id: 2, name: "Pete"},
  {id: 3, name: "Mary"}
];

let user = users.find(item => item.id == 1);
console.log(user.name) // John
```

array - special data structure to store ordered collection



array - special data structure to store ordered collection

reduce

- to iterate over an array – we can use forEach, for or for.of
- to iterate and return the data for each element – we can use map

The reduce() method

- φ executes a reducer function for array element.
- φ returns a single value: the function's accumulated result.
- φ does not execute the function for empty array elements.
- φ does not change the original array.

```
const numbers = [175, 50, 25];

function myFunc(total, num) {
  return total - num;
}
numbers.reduce (myFunc); //100
```

syntax

```
array.reduce (
      function (total, currentValue,
            currentIndex, arr),
      initialValue);
```

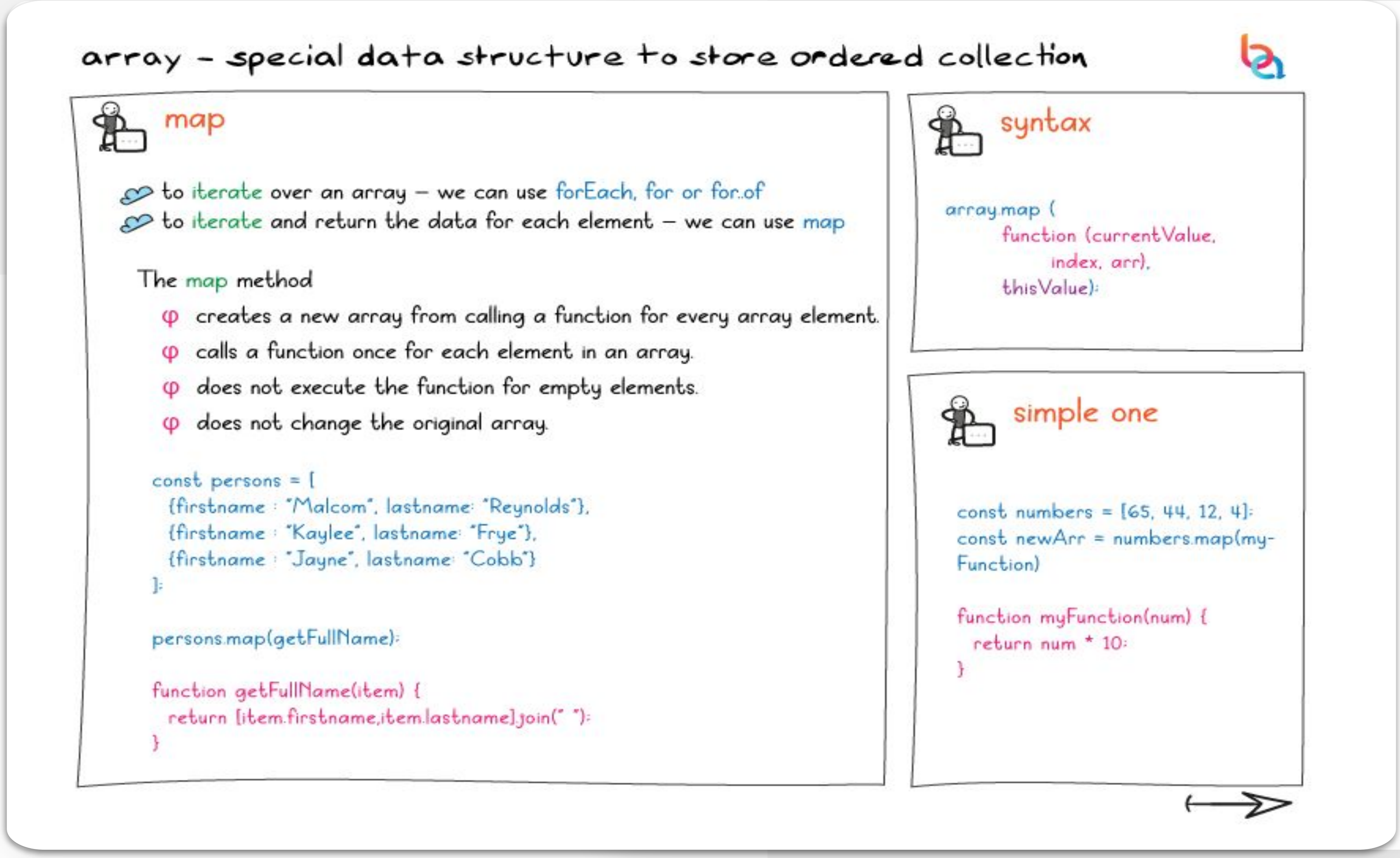
with initial value

```
const numbers = [15.5, 2.3, 1.1, 4.7];

numbers.reduce(getSum, 0);

function getSum(total, num) {
  return total + Math.round(num);
}
```

array - special data structure to store ordered collection



array - special data structure to store ordered collection

map

- to iterate over an array – we can use forEach, for or for.of
- to iterate and return the data for each element – we can use map

The map method

- φ creates a new array from calling a function for every array element.
- φ calls a function once for each element in an array.
- φ does not execute the function for empty elements.
- φ does not change the original array.

```
const persons = [
  {firstname : "Malcom", lastname: "Reynolds"},
  {firstname : "Kaylee", lastname: "Frye"},
  {firstname : "Jayne", lastname: "Cobb"}
];

persons.map(getFullName);

function getFullName(item) {
  return [item.firstname,item.lastname].join(" ");
}
```

syntax

```
array.map (
      function (currentValue,
            index, arr),
      thisValue);
```

simple one

```
const numbers = [65, 44, 12, 4];
const newArr = numbers.map(my-
Function)

function myFunction(num) {
  return num * 10;
}
```

array - special data structure to store ordered collection

Plan A : Practise

Plan B : Practise

References

[JS INFO]

[MDN JS Array]

Thank you

Hope you enjoyed the journey?

It’s now time for practise and keep visiting me for reference.