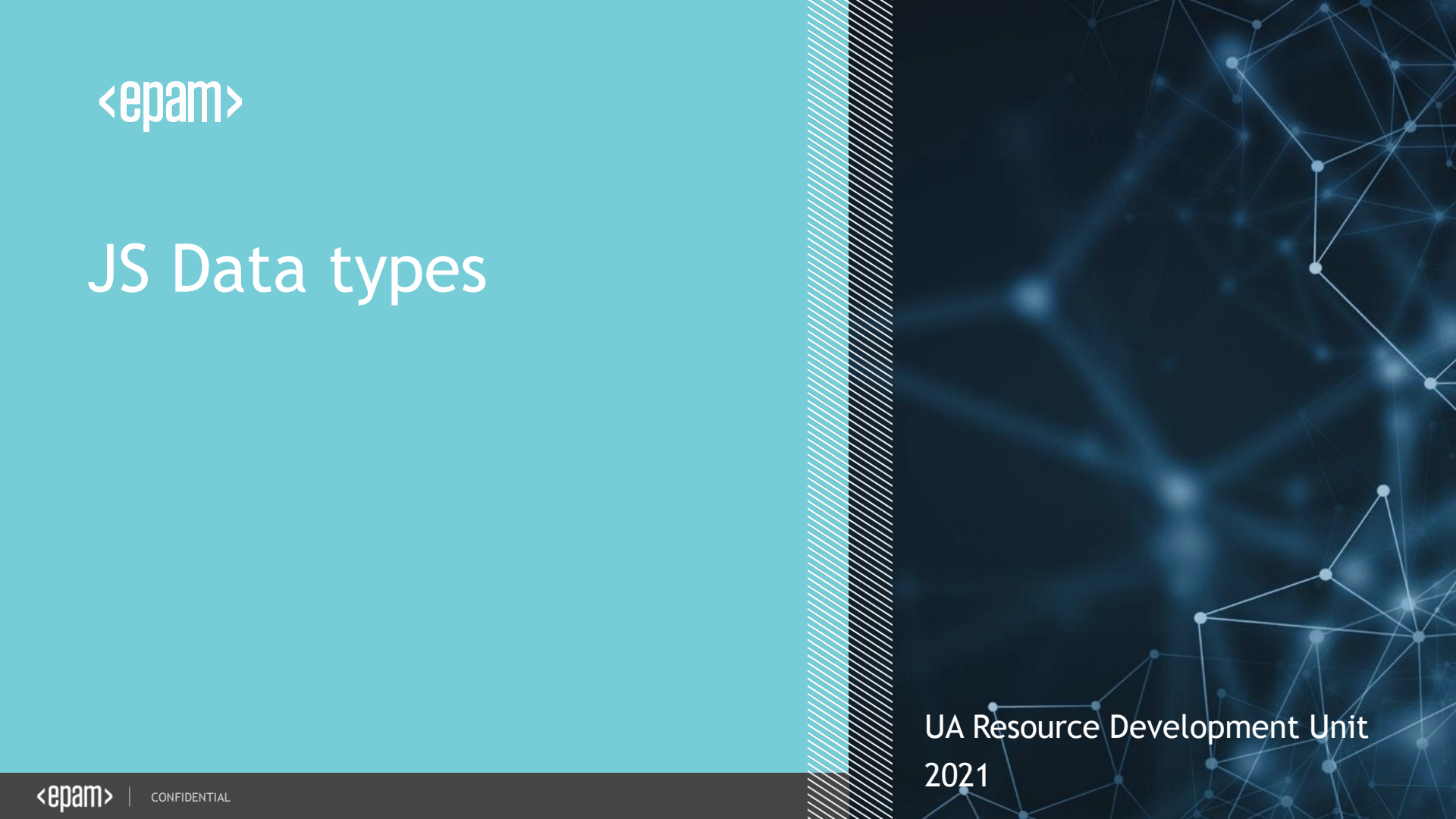

JS Data types
UA Resource Development Unit
2021

AGENDA

1. OBJECT TYPES
2. ARRAY
3. KEYED COLLECTIONS

OBJECT TYPES

OBJECT METHODS

Object static methods

- Object.keys()
- Object.getOwnPropertyNames()
- Object.freeze()
- Object.isFrozen()
- Object.assign() // ES6

EXAMPLES

Object.keys

- Returns an array of a given object's own enumerable properties, in the same order as that provided by a for...in loop.

JAVASCRIPT

```
let obj = { a: 1, b : 2, propName: 3 };

let keysArr = Object.keys(obj);


console.log(keysArr); // ["a", "b", "propName“]
```

Object.assign

- Method is used to copy the values of all enumerable own properties from one or more source objects to a target object. It will return the target object.

JAVASCRIPT

```
let initObj = { prop: 10 }, aObj = {};

let bObj = Object.assign({}, initObj);


Object.assign(aObj, initObj);
console.log(initObj);  //  { prop: 10 }
console.log(aObj);  //  { prop: 10 }
console.log(bObj);  //  { prop: 10 }
```

String

- let s='foo';
- let s="bar "
- let s = `\u00A9`
- var s = new String("foo");

var s1 = "2 + 2";

var s2 = new String("2 + 2");

eval(s1);

eval(s2);

```
var s = new String("foo");
console.log(s); // String {0: "f", 1: "o", 2: "o", length: 3,
[[PrimitiveValue]]: "foo"}
typeof s; //  'object'
```

String length

The length property has the string length

```
alert( `My\n`.length );
```

that `str.length`
is a numeric property, not a function. There is no need to add brackets after it.

Accessing characters

To get a character at position pos, use square brackets [pos] or call the method str.charAt(pos). The first character starts from the zero position

```
let str = `Hello`;

// the first character
alert(str[0]); // H
alert(str.charAt(0)); // H

// the last character
alert(str[str.length - 1]); // o
```

Strings are immutable

Strings can’t be changed in JavaScript. It is impossible to change a character.

```
let str = 'Hi';
str[0] = 'h'; // error
alert( str[0] ); // doesn't work
```

```
let str = 'Hi';
str = 'h' + str[1]; // replace the string
alert( str ); // hi
```

Searching for a substring

str.indexOf(substr, pos)

returns the position where the match was found or -1 if nothing can be found.

```
let str = "Widget";
if (~str.indexOf("Widget")) {
alert( 'Found it!' ); // works
}
```

```
let str = "Widget with id";
if (str.indexOf("Widget") != -1) {
alert("We found it"); // works now!
}
```

includes, startsWith, endsWith

includes()
method determines whether one string may be found within another string, returning true or false as appropriate.

Syntax
str.includes(*searchString*[, *position*])

```
var str = 'To be, or not to be, that is the question.';
console.log(str.includes('To be')); // true
console.log(str.includes('question')); // true
console.log(str.includes('nonexistent')); // false
console.log(str.includes('To be', 1)); // false
```

The methods str.startsWith and str.endsWith do exactly what they say:

```
alert( "Widget".startsWith("Wid") ); // true, "Widget" starts with "Wid"
alert( "Widget".endsWith("get") ); // true, "Widget" ends with "get
```

Getting a substring

method	selects…	negatives
slice(start, end)	from start to end	allows negatives
substring(start, end)	between start and end	negative values mean 0
substr(start, length)	from start get length characters	allows negative start

```
var str = 'Приближается утро.';
str.slice(-3);
str.slice(-3, -1);
str.slice(0, -1);

var anyString = 'Mozilla';
console.log(anyString.substring(-1, 3));
console.log(anyString.substring(3, 0));

var anyString = 'Mozilla';
var anyString4 = anyString.substring(anyString.length - 4);
console.log(anyString4);
```

```
var str = ‘abcdefg';

console.log('(1, 2): '   + str.substr(1, 2));
console.log('(-3, 2): '  + str.substr(-3, 2));
console.log('(-3): '     + str.substr(-3));
console.log('(1): '      + str.substr(1));
console.log('(-20, 2): ' + str.substr(-20, 2));
```

Correct comparisons

The **localeCompare()** method returns a number indicating whether a reference string comes before or after or is the same as the given string in sort order.

Syntax

referenceStr.localeCompare(*compareString*[, *locales*[, *options*]])

```
// The letter "a" is before "c" yielding a negative value
'a'.localeCompare('c'); // -2 or -1 (or some other negative value)

// Alphabetically the word "check" comes after "against" yielding a positive value
'check'.localeCompare('against'); // 2 or 1 (or some other positive value)

// "a" and "a" are equivalent yielding a neutral value of zero
'a'.localeCompare('a'); // 0
```

RegExp

```
var re = /ab+c/;
var re = new RegExp("ab+c");
```

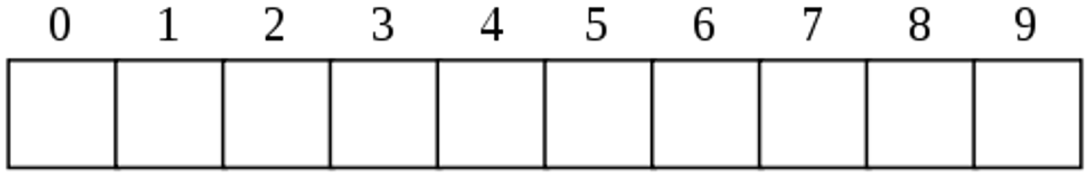
```
var re = /\w+\s/g;
var str = "fee fi fo fum";
var myArray = str.match(re);
console.log(myArray);
```

SUMMARY

- Strings in JavaScript are encoded using UTF-16.
- To get a character, use: [].
- To get a substring, use: slice or substring.
- To lowercase/uppercase a string, use: toLowerCase/toUpperCase.
- To look for a substring, use: indexOf, or includes/startsWith/endsWith for simple checks.
- To compare strings according to the language, use: localeCompare, otherwise they are compared by character codes.

Array

There exists a special data structure named Array, to store ordered collections.



Create an Array

```
let arr = new Array(element0, element1, ..., elementN);
let arr = Array(element0, element1, ..., elementN);
let arr = [element0, element1, ..., elementN];
```

```
let arr = new Array(1, true, ..., "Hello");
let arr = Array(1, true, ..., "Hello");
let arr = [1, true, ..., "Hello"];
```

Arrays in JS

```
1
2   let fruits = ['Apple', 'Banana']
3
4   console.log(fruits.length)
5   // 2
```

```
1
2   let first = fruits[0]
3   // Apple
4
5   let last = fruits[fruits.length - 1]
6   // Banana
```

Create an Array

```
var arr = [];
arr[3.4] = "Oranges";
console.log(arr);
console.log(arr.length);
```

```
var myArray = new Array("Hello", 6, 3.14159);
var myArray = ["Mango", "Apple", "Orange"]
```

Create an Array

Array.from()

```
alert(Array.from("str"));
```

Creates a new Array instance from an array-like or iterable object.

Create an Array

Array.of

```
var arr1 = new Array(2);
var arr2 = Array.of(2);
```

Creates a new Array instance with a variable number of arguments, regardless of number or type of the arguments.

fill()

arr.fill(*value*[, *start = 0[, end = this.length]]*)

```
[1, 2, 3].fill(4)          [4, 4, 4]
[1, 2, 3].fill(4, 1)       [1, 4, 4]
[1, 2, 3].fill(4, 1, 2)    [1, 4, 3]
```

The **`fill()`** method fills all the elements of an array from a start index to an end index with a static value.

length

```
var cats = ['Dusty', 'Misty', 'Twiggy'];
console.log(cats.length);

cats.length = 2;
console.log(cats);

cats.length = 0;
console.log(cats);

cats.length = 3;
console.log(cats);
```

```
3
(2) ["Dusty", "Misty"]

(0) []

(3) [undefined, undefined, undefined]
```

The **length** property of an object which is an instance of type Array sets or returns the number of elements in that array.

Spread in array literals

```
let array1 = [1,2,3];
let array2 = [4,...array1,5,6,7];
console.log(array2); // [4, 1, 2, 3, 5, 6, 7]
```

```
const arr = [4, 6, -1, 3, 10, 4];
const max = Math.max(...arr);
console.log(max);
```

Spread in array literals

```
let array1 = [1,2,3];
let array2 = [4,...array1,5,6,7];
console.log(array2); // [4, 1, 2, 3, 5, 6, 7]


let arr1 = [1];
let arr2 = [2];
let arr3 = [...arr1,...arr2,...[3,4]];
let arr4 = [5];


function mySum(a,b,c,d,e){
  return a+b+c+d+e;
}

let result = mySum(...arr3,...arr4);
console.log(result);
```

Destructuring assignment

```
var myArray = [1,2,3];
var a = myArray[0];
var b = myArray[1];
var c = myArray[2];
```

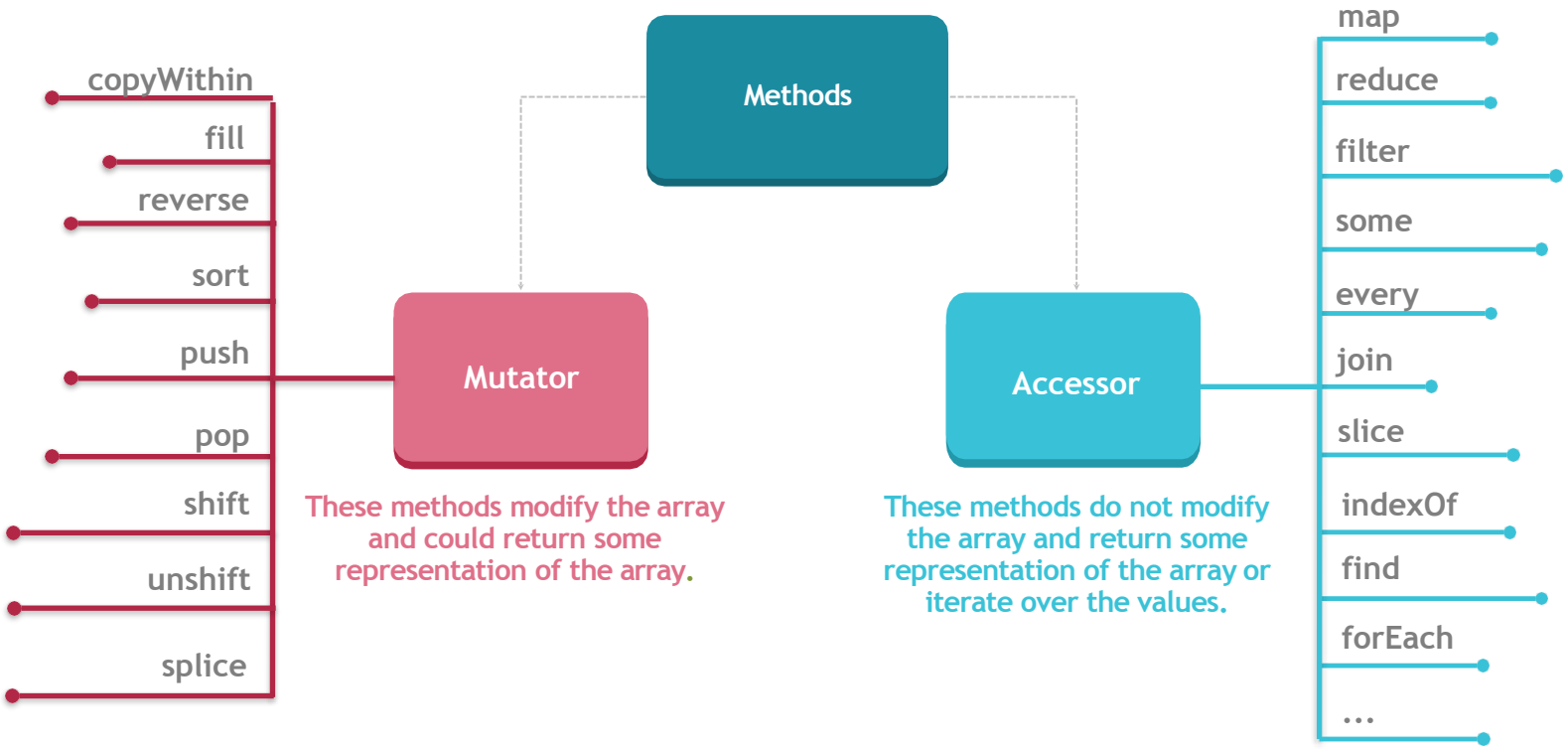
ES6:

```
let myArray = [1,2,3];
let a,b,c;
[a,b,c] = myArray;
```

```
let [a,b,c] =[1,2,3];
let [a, ,c] =[1,2,3];
let [a,b,c=3] =[1,2];
let [a,b,[c,d]] =[1,2,[3,4]];
let {a,b,c=3} = {a: '1', b: '2'}
```

Destructuring assignment is a special syntax that allows us to “unpack” arrays or objects into a bunch of variables

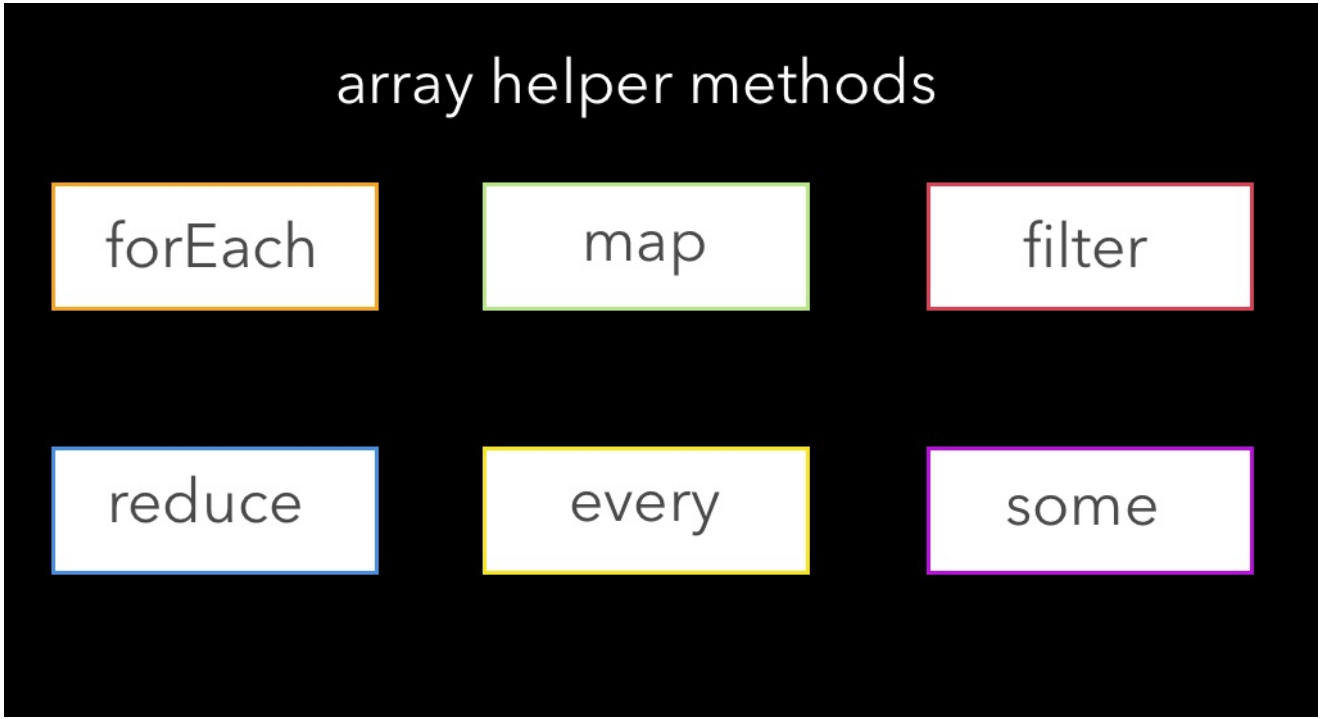
ARRAY PROTOTYPE METHODS



Array methods

ADDING ITEMS	`push()`	Adds one or more items to end of array and returns number of items in it
	`unshift()`	Adds one or more items to start of array and returns new length of it
REMOVING ITEMS	`pop()`	Removes last element from array (and returns the element)
	`shift()`	Removes first element from array (and returns the element)
ITERATING	`forEach()`	Executes a function once for each element in array*
	`some()`	Checks if some elements in array pass a test specified by a function*
	`every()`	Checks if all elements in array pass a test specified by a function*
COMBINING	`concat()`	Creates new array containing this array and other arrays/values
FILTERING	`filter()`	Creates new array with elements that pass a test specified by a function*
REORDERING	`sort()`	Reorders items in array using a function (called a compare function)
	`reverse()`	Reverses order of items in array
MODIFYING	`map()`	Calls a function on each element in array & creates new array with results

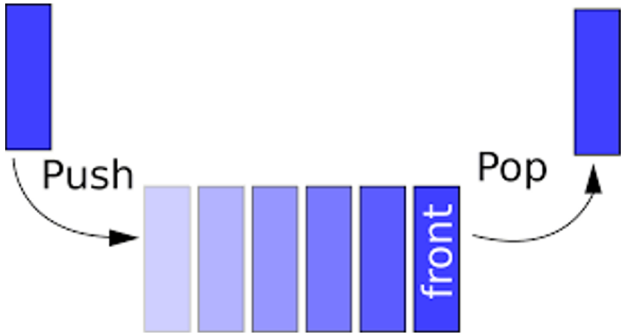
Array helper methods



MUTABLE METHODS

Methods pop/push, shift/unshift

A **queue** is one of most common uses of an array. In computer science, this means an ordered collection of elements which supports two operations:



push appends an element to the end.
shift get an element from the beginning, advancing the queue, so that the 2nd element becomes the 1st.

There's another use case for arrays – the data structure named **stack**.

push adds an element to the end.
pop takes an element from the end.
So new elements are added or taken always from the "end".

Methods pop/push, shift/unshift

pop
Extracts the last element of the array and returns it

Push
Append the element to the end of the array

Shift
Extracts the first element of the array and returns it

Unshift
Add the element to the beginning of the array

Methods `push/pop` run fast,
while `shift/unshift` are slow.

EXAMPLES

Array.prototype.push

- Method adds one or more elements to the end of an array and returns the new length of the array.

JAVASCRIPT

```
let arr = ["one", "two"];

arr.push(“three”); // 3

console.log(arr); // [“one“, “two“, “three"]
```

Array.prototype.pop

- Method removes the last element from an array and returns that element. This method changes the length of the array.

JAVASCRIPT

```
let arr = ["one", "two", "three"];

arr.pop(); // “three"

console.log(arr); // [“one“, “two"]
```

EXAMPLES

Array.prototype.splice

- Method changes the contents of an array by removing existing elements and/or adding new elements.

JAVASCRIPT

```
let arr = ["one", "two", "bad_value", "four"];

arr.splice(2, 1, "three");

console.log(arr); // ["one", "two", "three", "four"]
```

Array.prototype.fill

- Method fills all the elements of an array from a start index to an end index with a static value.

JAVASCRIPT

```
let arr = ["c", "b", "a"];

arr.fill("same");

console.log(arr); // ["same", "same", "same"]
```

IMMUTABLE METHODS

EXAMPLES

Array.isArray

- Method determines whether the passed value is an Array.

JAVASCRIPT

```
Array.isArray([1, 2, 3, "John"]);   // true

Array.isArray({ prop: 10 });        // false
```

Array.prototype.find (ES6)

- Method returns the value of the first element in the array that satisfies the provided testing function. Otherwise undefined is returned.

JAVASCRIPT

```
let arr = [1, 2, 28];

let foundValue = arr.find((value) => {
    return value % 14 === 0;
});  // 28

arr // [1, 2, 28]
```

loops

for

```
let arr = ["Apple", "Orange", "Pear"];
for (let i = 0; i < arr.length; i++) {
  alert( arr[i] );
}
```

for..of

```
let fruits = ["Apple", "Orange", "Plum"]; // iterates
over array elements
for (let fruit of fruits) {
  alert( fruit );
}
```

for .. in

```
let arr = ["Apple", "Orange", "Pear"];
for (let key in arr) {
  alert( arr[key] );
}
```

Cons

The loop `for..in` iterates over *all properties*, not only the numeric ones.

The `for..in` loop is optimized for generic objects, not arrays, and thus is 10-100 times slower.

Multidimensional arrays

Arrays can have items that are also arrays. We can use it for multidimensional arrays, to store matrices

```
let matrix = [
             [1, 2, 3],
             [4, 5, 6],
             [7, 8, 9]
];
alert( matrix[1][1] ); // the central element
```

summary

The `length` property is the array length or, to be precise, its last numeric index plus one. It is auto-adjusted by array methods.
`push(...items)` adds `items` to the end.
`pop()` removes the element from the end and returns it.
`shift()` removes the element from the beginning and returns it.
`unshift(...items)` adds items to the beginning.

To loop over the elements of the array:
`for (let i=0; i<arr.length; i++)` – works fastest, old-browser-compatible.
`for (let item of arr)` – the modern syntax for items only,
`for (let i in arr)` – never use.

Keyed collections

These objects represent collections which use keys; these contain elements which are iterable in the order of insertion.

Map

The **Map** object holds key-value pairs. Any value (both objects and primitive values) may be used as either a key or a value

```
let john = { name: "John" };
// for every user, let's store his visits count
let visitsCountMap = new Map();

// john is the key for the map
visitsCountMap.set(john, 123);
alert( visitsCountMap.get(john) );
```

The iteration goes in the same order as the values were inserted. Map preserves this order, unlike a regular Object.

Map can also use objects as keys.

Map Methods and properties

```
new Map() – creates the map.
map.set(key, value) – stores the value by the key.
map.get(key) – returns the value by the key, undefined if key doesn’t exist in map.
map.has(key) – returns true if the key exists, false otherwise.
map.delete(key) – removes the value by the key.
map.clear() – removes everything from the map.
map.size – returns the current element count.
```

For looping over a map:

map.keys() – returns an iterable for keys,
map.values() – returns an iterable for values,
map.entries() – returns an iterable for entries [key, value], it’s used by default in for..of.

Map Example

```
1
2   let map = new Map();
3
4   map.set('1', 'str1');   // a string key
5   map.set(1, 'num1');     // a numeric key
6   map.set(true, 'bool1'); // a boolean key
7
8   // remember the regular Object? it would convert keys to string
9   // Map keeps the type, so these two are different:
10  alert( map.get(1)   ); // 'num1'
11  alert( map.get('1') ); // 'str1'
12
13  alert( map.size ); // 3
```

Chaining

Every `map.set` call returns the map itself, so we can “chain” the calls:

```
map.set('1', 'str1')
map.set(1, 'num1')
map.set(true, 'bool1');
```

Set

The **Set** object lets you store unique values of any type, whether primitive values or object references.

Set objects are collections of values. You can iterate through the elements of a set in insertion order. A value in the Set **may only occur once**; it is unique in the Set's collection.

```
var mySet = new Set();
mySet.add(1); // Set { 1 }
mySet.add(5); // Set { 1, 5 }
mySet.add(5); // Set { 1, 5 }
```

Set Methods and properties

`new Set(iterable)` – creates the set, and if an iterable object is provided (usually an array), copies values from it into the set.
`set.add(value)` – adds a value, returns the set itself.
`set.delete(value)` – removes the value, returns true if value existed at the moment of the call, otherwise false.
`set.has(value)` – returns true if the value exists in the set, otherwise false.
`set.clear()` – removes everything from the set.
`set.size` – is the elements count.

For looping over a Set:
set.keys() – returns an iterable object for values,
set.values() – same as set.keys(), for compatibility with Map,
set.entries() – returns an iterable object for entries [value, value], exists for compatibility with Map.

Set Example

A Set is a special type collection – “set of values” (without keys), where each value may occur only once.

```
1
2   let set = new Set();
3
4   let john = { name: "John" };
5   let pete = { name: "Pete" };
6   let mary = { name: "Mary" };
7
8   // visits, some users come multiple times
9   set.add(john);
10  set.add(pete);
11  set.add(mary);
12  set.add(john);
13  set.add(mary);
14
15  // set keeps only unique values
16  alert( set.size ); // 3
17
18  for (let user of set) {
19    alert(user.name); // John (then Pete and Mary)
20  }
```

Q&A

UA Frontend Online LAB