Module 3

Web APIs and the `global` object

String & Number Methods

Array Methods

Object Methods and ES6 Modules

Array Methods

Array

The `Array` object, as with arrays in other programming languages, enables storing a collection of multiple items under a single variable name, and has members for performing common array operations.

Source

In JavaScript, arrays aren't primitives but are instead `Array` objects with the following core characteristics:

- **JavaScript arrays are resizable and can contain a mix of different data types**. (When those characteristics are undesirable, use typed arrays instead.)
- **JavaScript arrays are not associative arrays** and so, array elements cannot be accessed using arbitrary strings as indexes, but must be accessed using nonnegative integers (or their respective string form) as indexes.
- **JavaScript arrays are zero-indexed**: the first element of an array is at index `0`, the second is at index `1`, and so on — and the last element is at the value of the array's `length` property minus `1`.
- **JavaScript array-copy operations create shallow copies.** (All standard built-in copy operations with any JavaScript objects create shallow copies, rather than deep copies.)

Array Instance methods

Array.prototype.at()

Returns the array item at the given index. Accepts negative integers, which count back from the last item.

```
const array1 = [5, 12, 8, 130, 44];

let index = 2;

console.log(`An index of ${index} returns ${array1.at(index)}`);
// Expected output: "An index of 2 returns 8"

index = -2;

console.log(`An index of ${index} returns ${array1.at(index)}`);
// Expected output: "An index of -2 returns 130"
```

Array.prototype.concat()

Returns a new array that is the calling array joined with other array(s) and/or value(s).

```
const array1 = ['a', 'b', 'c'];
const array2 = ['d', 'e', 'f'];
const array3 = array1.concat(array2);

console.log(array3);
// Expected output: Array ["a", "b", "c", "d", "e", "f"]
```

Array.prototype.copyWithin()

Copies a sequence of array elements within an array.

```
const array1 = ['a', 'b', 'c', 'd', 'e'];

// Copy to index 0 the element at index 3
console.log(array1.copyWithin(0, 3, 4)); // target = 0, start = 3, end = 4
// Expected output: Array ["d", "b", "c", "d", "e"]

// Copy to index 1 all elements from index 3 to the end
console.log(array1.copyWithin(1, 3)); // target = 1, start = 3
// Expected output: Array ["d", "d", "e", "d", "e"]
```

Array.prototype.entries()

Returns a new *array iterator* object that contains the key/value pairs for each index in an array.

```
const array1 = ['a', 'b', 'c'];

const iterator1 = array1.entries();

console.log(iterator1.next().value);
// Expected output: Array [0, "a"]

console.log(iterator1.next().value);
// Expected output: Array [1, "b"]

console.log(iterator1.next().value);
// Expected output: Array [2, "c"]

console.log(iterator1.next().value);
// Expected output: undefined
```

Array.prototype.every()

Returns true if every element in the calling array satisfies the testing function.

```
const isBelowThreshold = (currentValue) => currentValue < 40;

const array1 = [1, 30, 39, 29, 10, 13];

console.log(array1.every(isBelowThreshold)); // callbackFn
// Expected output: true
```

Array.prototype.fill()

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

```
const array1 = [1, 2, 3, 4];

// Fill with 0 from position 2 until position 4
console.log(array1.fill(0, 2, 4)); // value, start, end
// Expected output: Array [1, 2, 0, 0]

// Fill with 5 from position 1
console.log(array1.fill(5, 1)); // value, start
// Expected output: Array [1, 5, 5, 5]

console.log(array1.fill(6)); // value
// Expected output: Array [6, 6, 6, 6]
```

Array.prototype.filter()

Returns a new array containing all elements of the calling array for which the provided filtering function returns `true`.

```
const words = ['spray', 'elite', 'exuberant', 'destruction', 'present'];

const result = words.filter((word) => word.length > 6);

console.log(result);
// Expected output: Array ["exuberant", "destruction", "present"]
```

Array.prototype.find()

Returns the value of the first element in the array that satisfies the provided testing function, or `undefined` if no appropriate element is found.

```
const array1 = [5, 12, 8, 130, 44];

const found = array1.find((element) => element > 10); // callbackFn

console.log(found);
// Expected output: 12
```

Array.prototype.findIndex()

Returns the index of the first element in the array that satisfies the provided testing function, or `-1` if no appropriate element was found.

```
const array1 = [5, 12, 8, 130, 44];

const isLargeNumber = (element) => element > 13;

console.log(array1.findIndex(isLargeNumber)); // callbackFn
// Expected output: 3
```

Array.prototype.findLast()

Returns the value of the last element in the array that satisfies the provided testing function, or `undefined` if no appropriate element is found.

```
const array1 = [5, 12, 50, 130, 44];

const found = array1.findLast((element) => element > 45); // callbackFn

console.log(found);
// Expected output: 130
```

Array.prototype.findLastIndex()

Returns the index of the last element in the array that satisfies the provided testing function, or `-1` if no appropriate element was found.

```
const array1 = [5, 12, 50, 130, 44];

const isLargeNumber = (element) => element > 45;

console.log(array1.findLastIndex(isLargeNumber)); // callbackFn
// Expected output: 3
// Index of element with value: 130
```

Array.prototype.flat()

Returns a new array with all sub-array elements concatenated into it recursively up to the specified depth.

```
const arr1 = [0, 1, 2, [3, 4]];

console.log(arr1.flat());
// expected output: Array [0, 1, 2, 3, 4]

const arr2 = [0, 1, [2, [3, [4, 5]]]];

console.log(arr2.flat());
// expected output: Array [0, 1, 2, Array [3, Array [4, 5]]]

console.log(arr2.flat(2)); // depth = 2
// expected output: Array [0, 1, 2, 3, Array [4, 5]]

console.log(arr2.flat(Infinity));
// expected output: Array [0, 1, 2, 3, 4, 5]
```

Array.prototype.flatMap()

Returns a new array formed by applying a given callback function to each element of the calling array, and then flattening the result by one level.

```
const arr1 = [1, 2, 1];

const result = arr1.flatMap((num) => (num === 2 ? [2, 2] : 1)); // callbackFn

console.log(result);
// Expected output: Array [1, 2, 2, 1]
```

Array.prototype.forEach()

Calls a function for each element in the calling array.

```
const array1 = ['a', 'b', 'c'];

array1.forEach((element) => console.log(element));

// Expected output: "a"
// Expected output: "b"
// Expected output: "c"
```

Array.prototype.includes()

Determines whether the calling array contains a value, returning `true` or `false` as appropriate.

```
const array1 = [1, 2, 3];

console.log(array1.includes(2));
// Expected output: true

const pets = ['cat', 'dog', 'bat'];

console.log(pets.includes('cat'));
// Expected output: true

console.log(pets.includes('at'));
// Expected output: false
```

Array.prototype.indexOf()

Returns the first (least) index at which a given element can be found in the calling array.

```
const beasts = ['ant', 'bison', 'camel', 'duck', 'bison'];

console.log(beasts.indexOf('bison'));
// Expected output: 1

// Start from index 2
console.log(beasts.indexOf('bison', 2));
// Expected output: 4

console.log(beasts.indexOf('giraffe'));
// Expected output: -1
```

Array.prototype.join()

Joins all elements of an array into a string.

```
const elements = ['Fire', 'Air', 'Water'];

console.log(elements.join());
// Expected output: "Fire,Air,Water"

console.log(elements.join(''));
// Expected output: "FireAirWater"

console.log(elements.join('-'));
// Expected output: "Fire-Air-Water"
```

Array.prototype.keys()

Returns a new *array iterator* that contains the keys for each index in the calling array.

```
const array1 = ['a', 'b', 'c'];
const iterator = array1.keys();

for (const key of iterator) {
  console.log(key);
}

// Expected output: 0
// Expected output: 1
// Expected output: 2
```

Array.prototype.lastIndexOf()

Returns the last (greatest) index at which a given element can be found in the calling array, or `-1` if none is found.

```
const animals = ['Dodo', 'Tiger', 'Penguin', 'Dodo'];

console.log(animals.lastIndexOf('Dodo'));
// Expected output: 3

console.log(animals.lastIndexOf('Tiger'));
// Expected output: 1

console.log(animals.lastIndexOf('Pallas cat'));
// Expected output: -1
```

Array.prototype.map()

Returns a new array containing the results of invoking a function on every element in the calling array.

```
const array1 = [1, 4, 9, 16];

// Pass a function to map
const map1 = array1.map((x) => x * 2); // callbackFn

console.log(map1);
// Expected output: Array [2, 8, 18, 32]
```

Array.prototype.pop()

Removes the last element from an array and returns that element.

```
const plants = ['broccoli', 'cauliflower', 'cabbage', 'kale', 'tomato'];

console.log(plants.pop());
// Expected output: "tomato"

console.log(plants);
// Expected output: Array ["broccoli", "cauliflower", "cabbage", "kale"]

plants.pop();

console.log(plants);
// Expected output: Array ["broccoli", "cauliflower", "cabbage"]
```

Array.prototype.push()

Adds one or more elements to the end of an array, and returns the new `length` of the array.

```
const animals = ['pigs', 'goats', 'sheep'];

const count = animals.push('cows');
console.log(count);
// Expected output: 4
console.log(animals);
// Expected output: Array ["pigs", "goats", "sheep", "cows"]

animals.push('chickens', 'cats', 'dogs');
console.log(animals);
// Expected output: Array ["pigs", "goats", "sheep", "cows", "chickens", "cats", "dogs"]
```

Array.prototype.reduce()

Executes a user-supplied "reducer" callback function on each element of the array (from left to right), to reduce it to a single value.

```
const array1 = [1, 2, 3, 4];

// 0 + 1 + 2 + 3 + 4
const initialValue = 0;
const sumWithInitial = array1.reduce(
  (accumulator, currentValue) => accumulator + currentValue, //callbackFn
  initialValue // initialValue (optional)
);

console.log(sumWithInitial);
// Expected output: 10
```

See details, next slide.

callbackFn

A function to execute for each element in the array. Its return value becomes the value of the `accumulator` parameter on the next invocation of `callbackFn`. For the last invocation, the return value becomes the return value of `reduce()`. The function is called with the following arguments:

- **`accumulator`**
  The value resulting from the previous call to `callbackFn`. On the first call, its value is `initialValue` if the latter is specified; otherwise its value is `array[0]`.
- **`currentValue`**
  The value of the current element. On the first call, its value is `array[0]` if `initialValue` is specified; otherwise its value is `array[1]`.
- **`currentIndex`**
  The index position of `currentValue` in the array. On the first call, its value is `0` if `initialValue` is specified, otherwise `1`.

Array.prototype.reduceRight()

Executes a user-supplied "reducer" callback function on each element of the array (from right to left), to reduce it to a single value.

```
const array1 = [
  [0, 1],
  [2, 3],
  [4, 5],
];

const result = array1.reduceRight((accumulator, currentValue) =>
  accumulator.concat(currentValue),
);

console.log(result);
// Expected output: Array [4, 5, 2, 3, 0, 1]
```

Array.prototype.reverse()

Reverses the order of the elements of an array in place. (First becomes the last, last becomes first.)

```
const array1 = ['one', 'two', 'three'];
console.log('array1:', array1);
// Expected output: "array1:" Array ["one", "two", "three"]

const reversed = array1.reverse();
console.log('reversed:', reversed);
// Expected output: "reversed:" Array ["three", "two", "one"]

// Careful: reverse is destructive -- it changes the original array.
console.log('array1:', array1);
// Expected output: "array1:" Array ["three", "two", "one"]
```

Array.prototype.shift()

Removes the first element from an array and returns that element.

```
const array1 = [1, 2, 3];

const firstElement = array1.shift();

console.log(array1);
// Expected output: Array [2, 3]

console.log(firstElement);
// Expected output: 1
```

Array.prototype.slice()

Extracts a section of the calling array and returns a new array.

```
const animals = ['ant', 'bison', 'camel', 'duck', 'elephant'];

console.log(animals.slice(2));
// Expected output: Array ["camel", "duck", "elephant"]

console.log(animals.slice(2, 4));
// Expected output: Array ["camel", "duck"]

console.log(animals.slice(1, 5));
// Expected output: Array ["bison", "camel", "duck", "elephant"]

console.log(animals.slice(-2));
// Expected output: Array ["duck", "elephant"]

console.log(animals.slice(2, -1));
// Expected output: Array ["camel", "duck"]

console.log(animals.slice());
// Expected output: Array ["ant", "bison", "camel", "duck", "elephant"]
```

Array.prototype.some()

Returns true if at least one element in the calling array satisfies the provided testing function.

```
const array = [1, 2, 3, 4, 5];

// Checks whether an element is even
const even = (element) => element % 2 === 0;

console.log(array.some(even));
// Expected output: true
```

Array.prototype.sort()

Sorts the elements of an array in place and returns the array.

If a **compare function** is omitted, the array elements are converted to strings, then sorted according to each character's Unicode code point value:

```
const months = ['March', 'Jan', 'Feb', 'Dec'];
months.sort();
console.log(months);
// Expected output: Array ["Dec", "Feb", "Jan", "March"]

const array1 = [1, 30, 4, 21, 100000];
array1.sort();
console.log(array1);
// Expected output: Array [1, 100000, 21, 30, 4]
```

We'll get back to making compare functions later.

Array.prototype.splice()

Adds and/or removes elements from an array.

```
const months = ['Jan', 'March', 'April', 'June'];
months.splice(1, 0, 'Feb');
// Inserts at index 1
console.log(months);
// Expected output: Array ["Jan", "Feb", "March", "April", "June"]

months.splice(4, 1, 'May');
// Replaces 1 element at index 4
console.log(months);
// Expected output: Array ["Jan", "Feb", "March", "April", "May"]
```

Array.prototype.toLocaleString()

Returns a localized string representing the calling array and its elements. Overrides the Object.prototype.toLocaleString() method.

```
const array1 = [1, 'a', new Date('21 Dec 1997 14:12:00 UTC')];
const localeString = array1.toLocaleString('en', { timeZone: 'UTC' });

console.log(localeString);
// Expected output: "1,a,12/21/1997, 2:12:00 PM"

const localeStringNo = array1.toLocaleString('no', { timeZone: 'CET' });

console.log(localeStringNo);
// Expected output: "1,a,21.12.1997, 15:12:00"
```

Array.prototype.toReversed()

Returns a new array with the elements in reversed order, **without modifying the original array**.

```
const items = [1, 2, 3];
console.log(items); // [1, 2, 3]

const reversedItems = items.toReversed();
console.log(reversedItems); // [3, 2, 1]
console.log(items); // [1, 2, 3]
```

Array.prototype.toSorted()

Returns a new array with the elements sorted in ascending order, **without modifying the original array**.

`toSorted()` also takes a Compare Function, like `sort()`:

```
const months = ["Mar", "Jan", "Feb", "Dec"];
const sortedMonths = months.toSorted();
console.log(sortedMonths); // ['Dec', 'Feb', 'Jan', 'Mar']
console.log(months); // ['Mar', 'Jan', 'Feb', 'Dec']

const values = [1, 10, 21, 2];
const sortedValues = values.toSorted((a, b) => a - b);
console.log(sortedValues); // [1, 2, 10, 21]
console.log(values); // [1, 10, 21, 2]
```

Array.prototype.toSpliced()

Returns a new array with some elements removed and/or replaced at a given index, **without modifying the original array**.

```
const months = ["Jan", "Mar", "Apr", "May"];

// Inserting an element at index 1
const months2 = months.toSpliced(1, 0, "Feb");
console.log(months2); // ["Jan", "Feb", "Mar", "Apr", "May"]

// Deleting two elements starting from index 2
const months3 = months2.toSpliced(2, 2);
console.log(months3); // ["Jan", "Feb", "May"]

// Replacing one element at index 1 with two new elements
const months4 = months3.toSpliced(1, 1, "Feb", "Mar");
console.log(months4); // ["Jan", "Feb", "Mar", "May"]

// Original array is not modified
console.log(months); // ["Jan", "Mar", "Apr", "May"]
```

Array.prototype.toString()

Returns a string representing the calling array and its elements. Overrides the `Object.prototype.toString()` method.

```
const array1 = [1, 2, 'a', '1a'];

console.log(array1.toString());
// Expected output: "1,2,a,1a"
```

Array.prototype.unshift()

Adds one or more elements to the front of an array, and returns the new `length` of the array.

```
const array1 = [1, 2, 3];

console.log(array1.unshift(4, 5)); // element1, element2
// Expected output: 5

console.log(array1);
// Expected output: Array [4, 5, 1, 2, 3]
```

Array.prototype.values()

Returns a new *array iterator* object that contains the values for each index in the array.

```
const array1 = ['a', 'b', 'c'];
const iterator = array1.values();

for (const value of iterator) {
  console.log(value);
}

// Expected output: "a"
// Expected output: "b"
// Expected output: "c"
```

Array.prototype.with()

Returns a new array with the element at the given index replaced with the given value, without modifying the original array.

```
const arr = [1, 2, 3, 4, 5];

console.log(arr.with(2, 6)); // [1, 2, 6, 4, 5]

console.log(arr); // [1, 2, 3, 4, 5]
```

“One-liners”

Array methods often allow you to significantly reduce your lines of code. Array methods can also be chained together.

This ultimately allows one to solve very complex issues with a single line of code, however, they are considered very difficult to read.

This also extends to using `for` loops and `if` statements with just one statement, i.e. without using a `{ }` to encapsule the code block.

You should try and avoid one-liners in the workplace, **when and if they reduce readability**, or at least make sure to document/comment what you are doing thouroughly (which kind of negate the point of the one-liner in the first place).

If you do choose to use one-liners in **a job interview**, be sure to mention that you are aware of issues with readability, but you are simply demonstrating that you have the capabilities of writing one-liners.

Examples of useful one-liners with bare minimum of comments:

```
/** shuffle a JavaScript array in random order */
const shuffleArray = (arr) => arr.sort(() => 0.5 - Math.random());

/** find the average of an array */
const calculateAverage = (arr) => arr.reduce((a, b) => a + b, 0) / arr.length;

/** Toggling a boolean value */
const toggle = (value) => value = !value

/** Remove all duplicate values from an array, using a Set */
const uniqueValues = (arr) => [...new Set(arr)]

/** Reverse a string */
const reverse = str => str.split('').reverse().join('');

/** Convert Celsius to Fahrenheit */
const celsiusToFahrenheit = (celsius) => celsius * 9/5 + 32;

/** Convert Fahrenheit to Celsius */
const fahrenheitToCelsius = (fahrenheit) => (fahrenheit - 32) * 5/9;

/** scroll to the top of a document */
const scrollToTop = () => window.scrollTo({top: 0, behavior: 'smooth'});
```

Todos

Mollify

Read Array Methods, and do the Lesson Task.