Array Methods

Prof. Andrew Sheehan

Boston University/Metropolitan College
Computer Science Department

<array>.forEach()

Iterates over each item in your array and invokes a function to handle it.

```
const ages = [15, 32, 28, 41, 67, 90];


ages.forEach(age => {
    // do something with the current value.
});
```

```
arr.forEach(function callback(currentValue, index, array) {
    //your iterator
}[, thisArg]);
```

Parameters

`callback`

Function to execute for each element, taking three arguments:

`currentValue`

The value of the current element being processed in the array.

`index`

The index of the current element being processed in the array.

`array`

The array that `forEach()` is being applied to.

`thisArg` | Optional

Value to use as **`this`** (i.e the reference `Object`) when executing `callback`.

Return value

`undefined`.

<array>.map()

Will create a new array, based on each value. The original array will not be affected.

```
const ages = [15, 30, 20, 41, 60, 80];

const agePlus7 = ages.map(age => return age + 7 );


      ages = [15, 30, 20, 41, 60, 80];  // unchanged..
agePlus7 = [22, 37, 27, 48, 67, 87];  // age +7 for each item
```

```
var new_array = arr.map(function callback(currentValue, index, array) {
    // Return element for new_array
}[, thisArg])
```

Parameters

`callback`

Function that produces an element of the new Array, taking three arguments:

`currentValue`

The current element being processed in the array.

`index`

The index of the current element being processed in the array.

`array`

The array `map` was called upon.

`thisArg`

Optional. Value to use as `this` when executing `callback`.

Return value

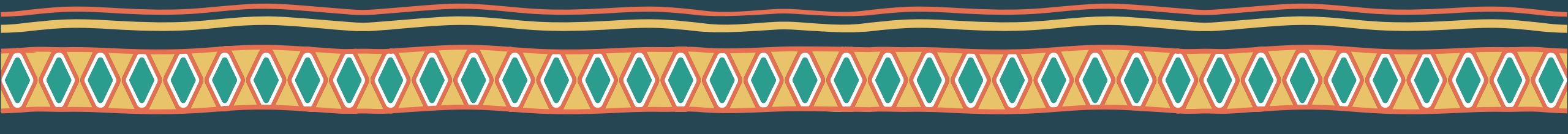
A new array with each element being the result of the callback function.

Using <array>.map() with built-in Object

```
const ages = [1, 4, 9, 16, 25, 36];

const ageSquared = ages.map(age => return Math.sqrt(age) );

// ageSqrt = [1, 2, 3, 4, 5, 6];
```

Using <array>.map() with Object literals

```
const clients = [
   { businessName: "Acme Inc.",  founder: "John Smith" },
   { businessName: "Microsoft Corp", founder: "Bill Gates" }
];

const businessOwner = clients.map(client => {
     return `${client.businessName}, ${client.founder}`;
});

// result: ["Acme Inc., John Smith", "Microsoft Corp, Bill Gates"]
```



<array>.reduce()

When you have an array of values and you want just one value.

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

const reducerFunction = function(accumulator, currentVal) {
    return accumulator + currentVal;
};

const sumOfAges = ages.reduce(reducerFunction, initialVal);
// 10
```

<array>.reduce()

Same thing, coded slightly more compact (less verbose).

```
const ages = [1,2,3,4].reduce( (sum, val) => { sum + val }, 0);
```

```
arr.reduce(callback[, initialValue])
```

Parameters

`callback`

Function to execute on each element in the array, taking four arguments:

`accumulator`

The accumulator accumulates the callback's return values; it is the accumulated value previously returned in the last invocation of the callback, or `initialValue`, if supplied (see below).

`currentValue`

The current element being processed in the array.

`currentIndex`

The index of the current element being processed in the array. Starts at index 0, if an `initialValue` is provided, and at index 1 otherwise.

`array`

The array `reduce` was called upon.

`initialValue`

[Optional] Value to use as the first argument to the first call of the `callback`. If no initial value is supplied, the first element in the array will be used. Calling reduce on an empty array without an initial value is an error.

<array>.find()

Finds the first match within your source array, If there is no match, undefined is returned. Does not affect the source array values.

```
const ages = [15, 32, 28, 41, 67, 90];

const result = ages.find(age => age > 21 );
// 32

const result = [15, 32, 28, 41, 67, 90].find( age =>  age > 21);

function first21 (age) { return age > 21 }

const result = ages.find(first21);
```

<array>.find() with Object literal

```
const market = [
   { name: ‘bananas’, quantity: 3400 },
   { name: ‘cherries’, quantity: 591},
   { name: ‘pears’, quantity: 39}
];

const result = market.find( data =>  data.name === ‘bananas’ )
// result = { name: “bananas”, quantity: 3400 }
```

Useful functions

Method	Finds
indexOf()	The value of the first element with a specified value
lastIndexOf()	The index of the last element with a specified value
find()	The value of the first element that passes a test
findIndex()	The index of the first element that passes a test
findLast()	The value of the last element that passes a test
findLastIndex()	The index of the last element that passes a test

<Array>.keys()

Returns an array of all the keys, per value in the array

```
var arr = ['a', 'b', 'c'];
console.log(Object.keys(arr)); // console: ['0', '1', '2']

// array like object
var obj = { 0: 'a', 1: 'b', 2: 'c' };
console.log(Object.keys(obj)); // console: ['0', '1', '2']

// array like object with random key ordering
var anObj = { 100: 'a', 2: 'b', 7: 'c' };
console.log(Object.keys(anObj)); // ['2', '7', '100']

// getFoo is property which isn't enumerable
var myObj = Object.create({}, {
  getFoo: {
    value: function () { return this.foo; }
  }
});
myObj.foo = 1;
console.log(Object.keys(myObj)); // console: ['foo']
```

<Array>.keys() – with empty values

Using Object.keys() will return only indices with values.

```
const names =  [“andrew”, , “joe”, ,“kathy”];
console.log( Object.keys(names));              // Array [“0”, “2”, “4”]
```

<Array>.includes()

Determines if an array has a target.

```
[1, 2, 3].includes(0);  // false
[1, 2, 3].includes(3); // true

[1, 2, 3].includes("3") // false
```

<Array>.values()

Returns only the values from an array

```
const names = ['andrew', 'joe', 'kathy'];
console.log( Object.values(names) );
//result:  Array ['andrew', 'joe', 'Kathy']

const iterator = names.values();
for (const val of iterator) { console.log(val) }

// "andrew"
// "joe"
// kathy
```