

# Methods on Arrays and Objects

Using array methods for cleaner JavaScript code

## JavaScript Fundamentals

Telerik Software Academy  
<https://telerikacademy.com>

Follow us



# Table of Contents

- Array methods for:
  - Conditions:
    - `Array#every`, `Array#some`
  - Transformation
    - `Array#filter`, `Array#reduce`, `Array#map`
  - Iteration
    - `Array#forEach`
  - Searching
    - `Array#find`, `Array#findIndex`
  - Others
    - `Array#fill`, `Array#sort`
- Polyfills for the missing array methods

Follow us



# Methods for Conditions

every, some

Follow us



# Array Methods for Conditions:

## Array#every

- Array#every
  - Signature: `[].every(callback);`
  - Callback: `callback(item [, index [, arr]])`
  - Returns: Boolean
  - Behavior: returns `true` if **ALL** the elements of the array meets the criteria in `callback()`
    - Returns `false` if **ANY** of the elements does not meet the criteria in `callback()`
  - Support: everywhere



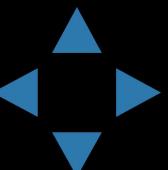
# Conditions: Array#every

- Examples:
  - Check if all the numbers in the array are odd?

```
function isOdd(number) {  
    return !(number % 2);  
}  
console.log([1, 2, 3, 4].every(isOdd)); //false  
console.log([1, 3, 5, 7].every(isOdd)); //true
```

- Check if all the numbers are greater than 18

```
function isGreaterThan18(number) {  
    return number > 18;  
}  
  
console.log([22, 23].every(isGreaterThan18)); //true  
console.log([19, 18].every(isGreaterThan18)); //false
```



# Array Methods for Conditions:

## Array#some

- Array#some
  - Signature: `[].some(callback);`
  - Callback: `callback(item [, index [, arr]])`
  - Returns: Boolean
  - Behavior: returns `true` if ANY of the elements of the array meets the criteria in `callback()`
    - Returns `false` if NONE of the elements meets the criteria in `callback()`
  - Support: everywhere

# Conditions: Array#some

- Examples:
  - Check if there is at least one odd number

```
function isOdd(number) {  
    return !(number % 2);  
}  
console.log([1, 2, 3, 4].some(isOdd)); //true  
console.log([1, 3, 5, 7].some(isOdd)); //true  
console.log([2, 4, 6, 8].some(isOdd)); //false
```

- Check if at least one number greater than 18

```
function isGreaterThan18(number) {  
    return number > 18;  
}  
console.log([22, 23].some(isGreaterThan18)); //true  
console.log([19, 18].some(isGreaterThan18)); //true  
console.log([17, 18].some(isGreaterThan18)); //false
```

# Array Methods for Transformation

map, reduce, filter

Follow us



# Array Methods for Transformation: Array#filter

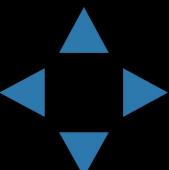
- Array#filter
  - Signature: `[ ].filter(callback);`
  - Callback: `callback(item [, index [, arr]])`
  - Returns: Array
  - Behavior: extracts in a new array only the elements that meet the criteria in `callback()`
    - Returns empty array, if no element meets the criteria
  - Support: everywhere

- Examples:
  - Extract the odd numbers from the array

```
function isOdd(number) {  
    return !(number % 2);  
}  
console.log([1, 2, 3, 4].filter(isOdd)); // [1, 3]  
console.log([1, 3, 5, 7].filter(isOdd)); // [1, 3, 5, 7]  
console.log([2, 4, 6, 8].filter(isOdd)); // []
```

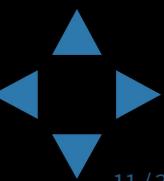
- Returns the numbers in a given range

```
function InRange(min, max) {  
    return function(item) { return min <= item && item <= max; };  
}  
var numbers = [2, 3, 4, 5, 6, 7, 8];  
console.log(numbers.filter(inRange(4, 7))); // [4, 5, 6, 7]  
console.log(numbers.filter(inRange(2, 4))); // [2, 3, 4]
```



# Array Methods for Transformation: Array#reduce

- Array#reduce
  - Signature: `[ ].reduce(callback, initial);`
  - Callback: `callback(accumulator, item [, index [, arr]])`
  - Returns: Object
  - Behavior: returns a single object, the result of the `callback()`
  - Support: everywhere

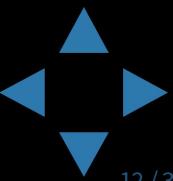


# Telerik Academy Transformations: Array#reduce

- Examples:
  - Calculates the sum and product on numbers

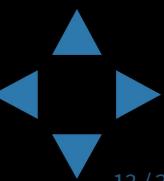
```
var sum = [1, 2, 3, 4].reduce(function(sum, number) {  
    return sum + number;  
}, 0);  
var product = [1, 2, 3, 4].reduce(function(sum, number) {  
    return sum * number;  
}, 1);  
console.log(sum);          //10  
console.log(product);      //24
```

Follow us



## Array#map

- **Array#map**
  - Signature: `[ ].map(callback);`
  - Callback: `callback(item [, index [, arr]])`
  - Returns: `Object`
  - Behavior: returns a new array with the same size. Each element is mapped, based on `callback()`
  - Support: everywhere

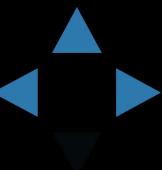


- Examples:
  - Calculates squares of an array of numbers

```
var squares = [1, 2, 3, 4, 5].map(function(number) {  
    return number * number;  
});  
console.log(squares); //prints [ 1, 4, 9, 16, 25 ]
```

- Parses a matrix given as an array of rows into an array of arrays of numbers

```
var lines = ['1 2 3',  
            '4 5 6'];  
var matrix = lines.map(function(line) {  
    return line.split(' ')  
        .map(Number);  
});  
console.dir(matrix);  
// [[1, 2, 3],  
//  [4, 5, 6]]
```



# Array Methods for Iteration

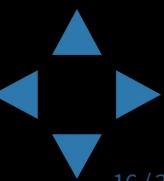
## forEach

Follow us



# Array Methods for Iteration: Array#forEach

- Array#forEach
  - Signature: `[ ].forEach(callback);`
  - Callback: `callback(item [, index [, arr]])`
  - Returns: `undefined`
  - Behavior: **iterates the elements** and passes each element as argument to callback
    - Much like a for-of loop where the callback is the body of the loop
  - Support: everywhere



# Iterations: Array#forEach

- Examples:
  - Print the elements of an array with their index

```
var numbers = ['One', 'Two', 'Three', 'Four', 'Five'];
numbers.forEach(function(item, index) {
    console.log('Item at' + index + 'has value' + item);
});
```

- Call a method for each object in an array

```
function createPerson(name, age) { //... }
var people = [createPerson('Peter', 13),
              createPerson('John', 18),
              createPerson('Susan', 21)];
people.forEach(function(person) {
    person.introduce();
});
```

# Array Methods for Searching

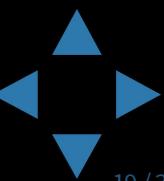
## find, findIndex

Follow us



# Array Methods for Searching: Array#find

- Array#find
  - Signature: `[ ].find(callback);`
  - Callback: `callback(item [, index [, arr]])`
  - Returns: Object or `undefined`
  - Behavior: returns the `leftmost element` in the array, that meets the criteria in `callback()`
    - If `no such element` is found, returns `undefined`
  - Support: Almost nowhere, needs a polyfill



## Searching: Array#find

- Examples
  - Find the leftmost odd number, greater than 5

```
var numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
console.log(numbers.find(function(item) {
    return !(item % 2) && item > 5;
})); //prints 7
```

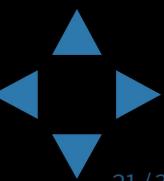
- Find the leftmost odd number, that is after index 3

```
var numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
console.log(numbers.find(function(item, index) {
    return index > 3 && !(item % 2);
})); //prints 5
```



# Array Methods for Searching: Array#findIndex

- Array#findIndex
  - Signature: `[ ].findIndex(callback);`
  - Callback: `callback(item [, index [, arr]])`
  - Returns: Number or -1
  - Behavior: returns the **index of the leftmost** element in the array, that **meets the criteria** in callback
    - If **no such element** is found, returns -1
  - Support: Almost nowhere, needs a polyfill

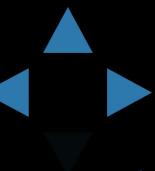


- Examples:
  - Find the index of leftmost odd number, greater than 5

```
var numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
console.log(numbers.findIndex(function(item) {
    return !(item % 2) && item > 5;
})); //prints 6(element 7)
```

- Find the index of the leftmost odd number, that is after index 3

```
var numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
console.log(numbers.findIndex(function(item, index) {
    return index > 3 && !(item % 2);
})); //prints 4(element 5)
```



# Other Array Methods

## sort, fill

Follow us



# Other Array Methods: Array#sort

- **Array#sort**
  - Signature: `[ ].sort(callback);`
  - Callback: `callback(obj1, obj2)`
  - Returns: `Array`
  - Behavior: sorts the items from the array, based on the `callback()`
  - Support: everywhere

Follow us

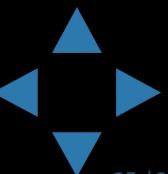


- Examples:
  - Sorts an array of numbers descending

```
var numbers = [5, 1, 2, 4, 6];
numbers.sort(function(x, y) {
    return y - x;
});
console.log(numbers);           // [ 6, 5, 4, 2, 1 ]
```

- Sorts array of people by name

```
var people = [createPerson('Peter', 13), createPerson('John', 18), ...];
people.sort(function(p1, p2) {
    return p1.name > p2.name;
});
console.log(people);           // John, Peter, Susan
```



## Other Array Methods: Array#fill

- Array#fill
  - Signature: `[ ].fill(value [, from [, to]])`
  - Returns: Array
  - Behavior: **fills an array** with the given value
  - Support: Almost nowhere, needs a polyfill

Follow us



# Array#fill

- Examples:
  - Fills an array with the number 1

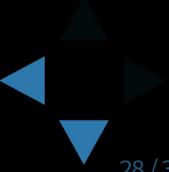
```
var count = 15,  
    arr = Array.from({length: count})  
      .fill(1);  
console.log(arr);
```

- Create array of arrays

```
var count = 5,  
    arr = Array.from({length: count}) // [ [ 1, 2, 3, 4, 5 ],  
      .fill([1, 2, 3, 4, 5]);           // [ 1, 2, 3, 4, 5 ],  
                                         // [ 1, 2, 3, 4, 5 ],  
                                         // [ 1, 2, 3, 4, 5 ],  
                                         // [ 1, 2, 3, 4, 5 ],  
                                         // [ 1, 2, 3, 4, 5 ] ]  
arr.fill([1, 2, 3, 4, 5]);  
console.log(arr);
```

# Chaining Methods

Follow us



# What is Chaining?

- Chaining is a pattern for calling/invokeing methods in functional programming
  - Each method returns an object
    - Another method can be called on this object
    - Etc...
- To implement chaining, follow the simple pattern:
  - If a method needs to return result -> Ok, return it
    - Else return an object to enable chaining

Follow us



# Chaining Array Methods

- Most of the array methods return a result
  - So they can be chained
  - *Example:*
    1. Fill an array with random digits
    2. Remove the even digits
    3. Return an array with the names of the remaining digits

- Solution:

```
var n = 10;
function getRandomDigit() { return (Math.random() * 10) | 0; }
function isOdd(item) { return !(item % 2); }
function digitToDigitName(digit) {
    return ['Zero', 'One', /* ... */, 'Nine'][digit];
}
var digitNames = Array.from({length: n})
    .fill(0)
    .map(getRandomDigit)
    .filter(isOdd)
    .map(digitToDigitName);
console.log(digitNames);
```

- Using ES 2015

```
var digitNames = digits.fill(0)
    .map(x => Math.random()*10 | 0)
    .filter(x=> !(x%2))
    .map(x=> [...][x]);
console.log(digitNames);
```

Follow us

