Array.prototype.some()

Summary: in this tutorial, you will learn how to use the JavaScript Array `some()` method to check if at least one element in the array passes a test.

Introduction to the JavaScript Array some() method

Sometimes, you want to check if an array has at least one element that meets a specified condition.

For example, to check if the following array has at least one element less than 5:

```
let marks = [ 4, 5, 7, 9, 10, 3 ];
```

...you typically use a `for` loop, like this:

```
let marks = [ 4, 5, 7, 9, 10, 3 ];

let lessThanFive = false;

for (let index = 0; index < marks.length; index++) {
    if (marks[index] < 5) {
        lessThanFive = true;
        break;
    }
}

console.log(lessThanFive);
```

Output:

```
true
```

How it works:

- First, declare a flag variable `lessThanFive` and set its value to `false` .
- Second, iterate over the elements. If an element is less than 5, set the flag to `true` and immediately exit the loop using the `break` statement.

The code works as expected. However, it is quite verbose.

The `Array` type provides an instance method `some()` that allows you to test if an array has at least one element that meets a condition.

```
let marks = [ 4, 5, 7, 9, 10, 3 ];

lessThanFive = marks.some(function(e) {
    return e < 5;
});

console.log(lessThanFive);
```

Output

```
true
```

The condition is implemented via a callback function passed into the `some()` method.

Now, the code is shorter. To make it more expressive, you can use the arrow function syntax in ES6:

```
let marks = [ 4, 5, 7, 9, 10, 3 ];

let lessThanFive = marks.some(e => e < 5);

console.log(lessThanFive);
```

JavaScript Array some() syntax

The following illustrates the syntax of the `some()` method:

```
arrayObject.some(callback[, thisArg]);
```

The `some()` method accepts two arguments:

1) The callback argument

The `some()` function executes the `callback` function once for each element in the array until it finds the one where the `callback` function returns a `true`. The `some()` method immediately returns `true` and doesn't evaluate the remaining elements.

If no element causes the `callback()` to return `true`, the `some()` method returns `false`.

The `callback` function takes three arguments:

```
function callback(currentElement [[, currentIndex], array]){ // ...}
```

- The `currentElement` is the current element being processed in the array.
- The `currentIndex` is the index of the current element being processed in the array.
- The `array` is array that `some()` was called upon.

2) The thisArg argument

The `thisArg` argument is optional. If you pass the `thisArg` into the method, you can use the `thisArg` as the `this` value inside the `callback` function.

JavaScript Array some() examples

Let's take some more examples of using the `some()` method.

1) Check if an element exists in the array

The following `exists()` function uses the `some()` method to check if a value exists in an array:

```
function exists(value, array) {
    return array.some(e => e === value);
}
```

```
let marks = [4, 5, 7, 9, 10, 2];

console.log(exists(4, marks));
console.log(exists(11, marks));
```

Output:

```
true
false
```

2) Check if an array has one element that is in a range

The following example shows how to check if any number in the `marks` array is in the range of (8, 10):

```
let marks = [4, 5, 7, 9, 10, 2];

const range = {
    min: 8,
    max: 10
};

let result = marks.some(function (e) {
    return e >= this.min && e <= this.max;
}, range);

console.log(result);
```

Output:

```
true
```

How it works.

- First, define a range object with min and max properties.

- Second, call the `some()` method on the marks array object and pass the callback and range object. Because we pass the range object as the second argument (`thisArg`), we can reference it inside the callback via the `this` value.

Notice that if you use the arrow function in this example, the `this` value inside the callback function does not bind to the `range` object but the `global` object.

Caution: Empty arrays

If you call the `some()` method on an empty array, the result is always `false` regardless of any condition. For example:

```
let result = [].some(e => e > 0);
console.log(result);

result = [].some(e => e <= 0);
console.log(result);
```

Output:

```
false
false
```

Summary

- Use the JavaScrip Array `some()` method to test if an array has at least one element that meets a condition.