# 6. Methods

1. Arrays

**array.concat(item...)**

Concat produces a new array containing a shallow copy of *this* array with the *items* appended to it. If an *item* is an array, then each of its elements is appended individually.

var a = ['a', 'b', 'c'];

var b = ['x', 'y', 'z'];

var c = a.concat(b, true);

// c is ['a', 'b', 'c', 'x', 'y', 'z', true]

**array.join(separator)**

Join method makes a string from an array. It does this by making a string of each of the array’s elements, and then concatenating them all together with a separator between them. The default separator is ','. To join without separation, use an empty string as the separator.

var a = ['a', 'b', 'c'];

var c = a.join(''); // c is 'abc';

**array.pop( )**

Pop method removes and returns the last element in this array. If the array is empty, it returns undefined.

var a = ['a', 'b', 'c'];

var c = a.pop(); // a is ['a', 'b'] & c is 'c';

**array.push(item...)**

Push method appends items to the end of an array. Unlike the concat method, it modifies the array and appends array items whole. It returns the new length of the array:

var a = ['a', 'b', 'c'];

var b = ['x', 'y', 'z'];

var c = a.push(b, true);

// a is ['a', 'b', 'c', ['x', 'y', 'z'], true]

// c is 5;

**array.reverse( )**

The reverse method modifies the array by reversing the order of the elements. It returns the array:

var a = ['a', 'b', 'c'];

var b = a.reverse();

// Both **a** and **b** are ['c', 'b', 'a']

**array.shift( )**

The shift method removes the first element from an array and returns it. If the array is empty, it returns *undefined*. Shift is usually much slower than pop:

var a = ['a', 'b', 'c'];

var c = a.shift(); // a is ['b', 'c'] & c is 'a'

**array.slice(start, end)**

Slice makes a shallow copy of a portion of an array. The first element copied will be array[*start*] and It will stop before copying array[*end*]. The end parameter is optional, and the default is array.length. If either parameter is negative, array.length will be added to them in an attempt to make them nonnegative. If start is greater than or equal to array.length, the result will be a new empty array.

var a = ['a', 'b', 'c'];

var b = a.slice(0, 1); // b is ['a']

var c = a.slice(1); // c is ['b', 'c']

var d = a.slice(1, 2); // d is ['b']

var e = a.slice(a.length, a.length); // e is []

**array.sort(comparefn)**

The sort method sorts the contents of an array in place.

var m = ['aa', 'bb', 'a', 4, 8, 15, 16, 23, 42];

m.sort(function (a, b) {

if (a === b) {

return 0;

}

if (typeof a === typeof b) {

return a < b ? -1 : 1;

}

return typeof a < typeof b ? -1 : 1;

});

// m is [4, 8, 15, 16, 23, 42, 'a', 'aa', 'bb']

If case is not significant, your comparison function should convert the operands to lower-case before comparing them.

**array.splice(start, deleteCount, item...)**

The splice method removes elements from an array, replacing them with new items. The start parameter is the number of a position within the array. The deleteCount parameter is the number of elements to delete starting from that position. If there are additional parameters, those items will be inserted at the position. It returns an array containing the deleted elements.

var a = ['a', 'b', 'c'];

var r = a.splice(1, 1, 'ache', 'bug');

// a is ['a', 'ache', 'bug', 'c']

// r is ['b']

**array.unshift(item...)**

The unshift method is like the push method except that it shoves the items onto the front of this array instead of at the end. It returns the array’s new length:

var a = ['a', 'b', 'c'];

var r = a.unshift('?', '@');

// a is ['?', '@', 'a', 'b', 'c']

// r is 5

**function.apply(thisArg, argArray)**

The apply method invokes a function, passing in the object that will be bound to *this* and an optional array of arguments.

var arr = ['1px','solid', 'white'];

var Obj = function(border, type, color){

this.border = border;

this.type = type;

this.color = color;

}

Obj.prototype.kind = function(a, b, c){

console.log(

this.border + " " + a,

this.type + " " + b,

this.color + " " + c

)

}

var a = {

border: '2',

type: '4',

color: '6',

}

var foo = new Obj(1,3,4);

Obj.prototype.kind.apply(a, arr) // "2 1px" "4 solid" "6 white"

foo.kind.apply(a, arr) // "2 1px" "4 solid" "6 white"

**function.call(thisArg, arg1, arg2…)**

The call method invokes a function, passing in the object that will be bound to *this* and optional comma-separated arguments.

**function.bind(thisArg, arg1, arg2…)**

The call method copies a function, passing in the object that will be bound to *this* and optional comma-separated arguments.

**number.toExponential(fractionDigits)**

The toExponential method converts this number to a string in the exponential form. The optional fractionDigits parameter controls the number of decimal places. It should be between 0 and 20:

document.writeln(Math.PI.toExponential(0)); //3e+0 document.writeln(Math.PI.toExponential(2)); //3.14e+0 document.writeln(Math.PI.toExponential(7)); //3.1415927e+0

document.writeln(Math.PI.toExponential(16)); //3.1415926535897930e+0

**number.toFixed(fractionDigits)**

The toFixed method converts this number to a string in the decimal form. The optional fractionDigits parameter controls the number of decimal places. It should be between 0 and 20. The default is 0:

document.writeln(Math.PI.toFixed(0)); // 3

document.writeln(Math.PI.toFixed(2)); // 3.14

document.writeln(Math.PI.toFixed(7)); // 3.1415927

**number.toPrecision(precision)**

The toPrecision method converts this number to a string in the decimal form. The optional precision parameter controls the number of digits of precision. It should be between 1 and 21:

document.writeln(Math.PI.toPrecision(2)); // 3.1 document.writeln(Math.PI.toPrecision(7)); // 3.141593 document.writeln(Math.PI.toPrecision(16)); // 3.141592653589793 document.writeln(Math.PI.toPrecision( )); // 3.141592653589793

**number.toString(radix)**

The toString method converts this number to a string. The optional radix parameter controls radix, or base. It should be between 2 and 36. The default radix is base 10.

The most common case, number.toString( ), can be written more simply as String(number):

document.writeln(Math.PI.toString( )); // 3.141592653589793