Mastering Javascript Arrays

By Patrick Hunlock

Believe it or not, the very first version of Javascript shipped without Arrays. Subsequent versions made up for the oversight and modern Javascript Arrays are powerful structures indeed, even emulating many common data structures such as stacks and queues. This reference will cover the core functionality of Arrays as well as introduce a few useful extensions.

Introduction

An Array is an enumerated list of variables. It's a programming construct that allows programmers to replace this…

```
x0=0; x1=1; x2=2; x3=3; x4=4; x5=5;
```

…with this…

```
x[0]=0; x[1]=1; x[2]=2; x[3]=3; x[4]=4; x[5]=5;
```

The index (the number in the brackets `[]`)can be referenced by a variable, allowing for easy looping through the data structure.

```
for(i=0; i<6; i++) {
   document.writeln(x[i]+', ');
}
```

Which will output the following…

```
0, 1, 2, 3, 4, 5
```

Creating A New Array

Most tutorials start out introducing you to arrays as such…

```
var myArray = new Array(10);
```

Current best-practice eschews the new keyword on Javascript primitives. If you want to create a new Array simply use brackets `[]` like this…

```
var myArray = [];
```

You don't need to tell Javascript how many items to size the Array for. Javascript will automatically increase the size of the Array as needed, as you add items into the Array. Creating an Array with brackets instead of with the new constructor avoids a bit of confusion where you want to initialize only one integer. For instance...

```
var badArray =new Array(10); // Creates an empty Array that's sized for 10 elements.
var goodArray=[10];          // Creates an Array with 10 as the first element.
```

As you can see these two lines do two very different things. If you had wanted to add more than one item then `badArray` would be initialized correctly since Javascript would then be smart enough to know that you were initializing the array instead of stating how many elements you wanted to add.

Since the new constructor is not necessary with Arrays and there's a slight chance of unintended results by using the new constructor, it's recommended you not use new `Array()` to create an Array.

Initializing An Array

You can initialize your array with predefined data...

```
var myArray = ['January', 'February', 'March'];

document.writeln('0>'+myArray[0]+'<BR>');       // Will output: 0>January
document.writeln('1>'+myArray[1]+'<BR>');       // Will output: 1>February
document.writeln('2>'+myArray[2]+'<BR>');       // Will output: 2>March
```

You can initialize your array with data after an empty array has been created...

```
var myArray = [];
myArray[0] = 'January';
myArray[1] = 'February';
myArray[2] = 'March';
document.writeln('0>'+myArray[0]+'<BR>'); // Will output: 0>January
document.writeln('1>'+myArray[1]+'<BR>'); // Will output: 1>February
document.writeln('2>'+myArray[2]+'<BR>'); // Will output: 2>March
```

If you skip an element, the blank Array elements will be of type `undefined`...

```
var myArray = [];
myArray[0] = 'January';
myArray[1] = 'February';
myArray[5] = 'March';
document.writeln('0>'+myArray[0]+'<BR>'); // Will output: 0>January
document.writeln('1>'+myArray[1]+'<BR>'); // Will output: 1>February
document.writeln('2>'+myArray[2]+'<BR>'); // Will output: 2>undefined
document.writeln('3>'+myArray[3]+'<BR>'); // Will output: 3>undefined
document.writeln('4>'+myArray[4]+'<BR>'); // Will output: 4>undefined
document.writeln('5>'+myArray[5]+'<BR>'); // Will output: 5>March
```

Storing Data In An Array

An array can store anything you can assign to a variable: booleans, numbers, strings, functions, objects, other Arrays, even regular expressions...

```
var myArray=[3,'hello!',function() {return 5},{'color':'blue','budget':25},/[ell]/i ];
document.writeln('0>'+myArray[0]+'<BR>');                  // Will output: 0>3
document.writeln('1>'+myArray[1]+'<BR>');                  // Will output: 1>hello!
document.writeln('2>'+myArray[2]()+'<BR>');                // Will output: 2>5
document.writeln('3>'+myArray[3].color+'<BR>');            // Will output: 3>blue
document.writeln('3>'+myArray[3].budget+'<BR>');           // Will output: 3>25
document.writeln('4>'+myArray[4].test(myArray[1])+'<BR>'); // Will output: 4>true
```

Multi-Dimensional Arrays

Since an Array can store other Arrays you can get the benefit of multi-dimension arrays.

```
var x=[0,1,2,3,4,5];
var y=[x];
```

In the above example we created an array named x and assigned it as the first element in the array y. If we ask for the value of `y[0]` it will return the contents of `x` as a string because we didn't specify an index.

```
var x=[0,1,2,3,4,5];
var y=[x];
document.writeln(y[0]); // Will output: 0,1,2,3,4,5
```

If we wanted the third index we'd access it this way...

```
var x=[0,1,2,3,4,5];
var y=[x];
document.writeln(y[0][3]); // Will output: 2
```

There's no defined limit to how many Arrays you can nest in this manner. For instance ...

```
document.writeln(bigArray[5][8][12][1])
```

...would indicate `bigArray`'s 5th index held an array, who's 8th index held an array, who's 12th index held an array, who's first index contains the data we want.

JavaScript Arrays Are Passed By Reference

Arrays are passed to functions by reference, or as a pointer to the original. This means anything you do to the Array inside the function affects the original.

```
var myArray = [ 'zero', 'one', 'two', 'three', 'four', 'five' ];
document.writeln(myArray[1]); // Will output: one
function passedByReference(refArray) {
   refArray[1] = 'changed';
}
passedByReference(myArray);
document.writeln(myArray[1]); // Will output: changed
```

JavaScript Arrays Are Assigned By Reference

Assigning an Array to a new variable creates a pointer to the original Array. For instance…

```
var myArray = [ 'zero', 'one', 'two', 'three', 'four', 'five' ];
var newArray= myArray;
newArray[1] = 'changed';
document.writeln(myArray[1]); // Will output: changed
```

Passing Arrays As Values

To pass an Array by ***value*** instead of by reference, use the `Array.slice()` method.

```
var myArray = [ 'zero', 'one', 'two', 'three', 'four', 'five' ];
var newArray= myArray.slice();
newArray[1] = 'changed';
document.writeln(myArray[1]); // Will output: one
function passedByReference(refArray) {
   refArray[1] = 'changed';
}
passedByReference(myArray.slice());
document.writeln(myArray[1]); // Will output: one
```

Array.length

Every Array has a `length` property. This always contains the number of elements in the array. Since Arrays always start at zero, the length property is convenient for loops since it will always be one greater than the actual index. For instance if the Array has 10 elements then the indexes will be 0-9, so as long as our counter is less than the `Array.length` we'll cover the entire Array...

```
for (var i=0; i<myArray.length; i++) {}
```

Going back to our undefined example above. Even though 3 of the Array items are undefined the length property will still count them because it's always one higher than the highest accessible index value.

```
var myArray = [];
myArray[0] = 'January';
myArray[1] = 'February';
myArray[5] = 'March';
document.writeln('0>'+myArray[0]+'<BR>');          // Outputs 0>January
document.writeln('1>'+myArray[1]+'<BR>');          // Outputs 1>February
document.writeln('2>'+myArray[2]+'<BR>');          // Outputs 2>undefined
document.writeln('3>'+myArray[3]+'<BR>');          // Outputs 3>undefined
document.writeln('4>'+myArray[4]+'<BR>');          // Outputs 4>undefined
document.writeln('5>'+myArray[5]+'<BR>');          // Outputs 5>March
document.writeln('Array Length: '+myArray.length); // Outputs Array Length: 6
```

`Array.length` is **NOT** a read-only value -- you can set it as you wish. If you have 100 elements in an array and set the length to 50, Javascript will truncate the last 50 elements from the array (effectively deleting them). If you have 10 elements in an array and set `Array.length` to 100 then the length of the array will be expanded to 100, creating 90 undefined elements after the original 10 items.

Javascript Does Not Support Associative Arrays

An associative array is an array which uses a string instead of a number as an index.

```
var normalArray    = [];
normalArray[1] = 'This is an enumerated array';
alert(normalArray[1]);                    // Outputs: This is an enumerated array
var associativeArray           = [];
associativeArray['person'] = 'John Smith';
alert(associativeArray['person']);        // Outputs: John Smith
```

Javascript does not have, and does not support Associative Arrays. However... All arrays in Javascript are objects and Javascript's object syntax gives a basic emulation of an associative Array. For this reason the example code above will actually work. Be warned that this is not a real array and it has real pitfalls if you try to use it. The 'person' element in the example becomes part of the Array object's properties and methods, just like `.length`, `.sort()`, `.splice()`, and all the other built-in properties and methods.

You can loop through an object's properties with the following syntax...

```
var associativeArray = [];
associativeArray["one"] = "First";
associativeArray["two"] = "Second";
associativeArray["three"] = "Third";
for (i in associativeArray) {
   document.writeln(i+':'+associativeArray[i]+', ');
   // outputs: one:First, two:Second, three:Third
};
```

In the above example, `associativeArray.length` will be zero because we didn't actually put anything into the Array, we put it into a`ssociativeArray's` object. `associativeArray[0]` will be undefined.

The loop in the above example will also pick up any methods, properties, and prototypes which have been added to the array and not just your data. A lot of problems people have with the Prototype framework is that their associative arrays break because Prototype adds a few useful functions to the global object and `for i in x` loops pick up those additional methods. That's the pitfall of using Array/objects as a poor man's associative array.

As a final example, the previous code will work regardless of whether you define `associativeArray` as an `Array` (`[]`), an `Object`(`{}`), a `Regular Expression` (`//`), `String`(`""`), or any other Javascript object.

The bottom line is -- don't try to use associative arrays. Code for what they are -- object properties, not Arrays.

Array Methods Reference

Since Javascript Arrays are modified objects, each and every Array you create has a few core methods. What's really interesting is that some of these methods implement basic data structures you'd normally have to write yourself such as stacks (`push, pop`) and queues (`shift, unshift`).

Method	IE Version	Mozilla Version	Notes
`concat`	4.0	4.0	Joins multiple Arrays/Values.
`every`	*	FF 1.5	Calls a function for every element of the array until false is returned.
`filter`	*	FF 1.5	Creates an array with each element which evaluates true in the function provided.
`forEach`	*	FF 1.5	Executes a specified function on each element of an Array
`join`	3.0	3.0	Joins all the Array elements together into a string.
`indexOf`	*	FF 1.5	Searches the Array for specific elements.
`lastIndexOf`	*	FF 1.5	Returns the last item in the Array which matches the search critera.
`map`	*	FF 1.5	Creates a new array with the result of calling the specified function on each element of the Array.
`pop`	5.5	4.0	Returns the last item in the Array and removes it from the Array.
`push`	5.5	4.0	Adds the item to the end of the Array.
`reverse`	3.0	3.0	Reverses the Array so the last item becomes the first and vice-versa.
`shift`	5.5	4.0	Returns the first item in the Array and removes it from the Array.
`slice`	4.0	4.0	Returns a new array from the specified index and length.
`some`	*	FF 1.5	Passes each element through the supplied function until true is returned.
`sort`	3.0	3.0	Sorts the array alphabetically or by the supplied function.
`splice`	5.5	4.0	Deletes the specified index(es) from the Array.
`toSource`	**	FF 1.5	Returns the source code of the array.
`toString`	3.0	3.0	Returns the Array as a string.
`unshift`	5.5	4.0	Inserts the item(s) to the beginning of the Array.
`valueOf`	3.0	3.0	Like toString, returns the Array as a string.

* Prototype functions are available to make this method available to Internet Explorer and older browsers.

** Not supported.

Array.concat(value1[value2[value...]])

The `concat` method appends the passed values to the end of the Array, passing back a ***NEW*** array containing the joined values. The values passed to the `concat` method can be anything you can assign to a variable in Javascript.

```
var myArray = [1,2,3];
var newArray= [4,5,6];
var seven = 7;
var eight = 'eight';
var nine = {'sky':'blue', 'grass':'green'};
var joinedArray=myArray.concat(newArray, seven, eight, nine);
document.writeln(myArray);     // outputs: 1,2,3
document.writeln(joinedArray); // outputs: 1,2,3,4,5,6,7,'eight',[object Object]
```

Supported Since: Netscape 4.0, IE 4.0

Array.every(function)

The every method is a Firefox method which accepts a function as an argument. Every value of the array is passed to that function until the function returns false. If no elements return false then every will return true, if an element returned false then every will return false. It's a convenient way to test an Array and see if every element is a number for instance.

This method will pass the current value, the current index, and a pointer to the array to your function -- `myfunction(curValue, curIndex, curArray)`.

```
var isNumeric = function(x) {
   // returns true if x is numeric and false if it is not.
   var RegExp = /^(-)?(\d*)(\.?)(\d*)$/;
   return String(x).match(RegExp);
}
var myArray = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15];
document.writeln(myArray.every(isNumeric));   // outputs: true
var myArray = [1,2,3,4,5,6,7,8,9,'ten',11,12,13,14,15];
document.writeln(myArray.every(isNumeric));   // outputs: false
```

This method can be prototyped to allow Internet Explorer and older browsers to use this method. Simply copy the following code into your Javascript toolbox and the .every() method will be available regardless of your browser version.

```
//This prototype is provided by the Mozilla foundation and
//is distributed under the MIT license.
//http://www.ibiblio.org/pub/Linux/LICENSES/mit.license
if (!Array.prototype.every)
{
  Array.prototype.every = function(fun /*, thisp*/)
  {
    var len = this.length;
    if (typeof fun != "function")
      throw new TypeError();
    var thisp = arguments[1];
    for (var i = 0; i < len; i++)
    {
      if (i in this &&
          !fun.call(thisp, this[i], i, this))
        return false;
    }
    return true;
  };
}
```

Supported Since: Firefox 1.5, Internet Explorer: ---

Array.filter(function)

Filter creates a new Array of items which evaluate to true in the supplied function. In the `Array.every()` method, we tested if the entire Array was composed of Numbers. In `Array.filter()` we can extract all the numbers, creating a new Array in the process.

This method will pass the current value, the current index, and a pointer to the array to your function -- `myfunction(curValue, curIndex, curArray)`.

Here we pass the array through the same function as `.every()` -- `isNumeric` -- and if the element is a number it's placed in the new `oddArray` Array.

```
var isNumeric = function(x) {
   // returns true if x is numeric and false if it is not.
   var RegExp = /^(-)?(\d*)(\.?)(\d*)$/;
   return String(x).match(RegExp);
}
var myArray = [1,'two',3,'four',5,'six',7,'eight',9,'ten'];
var oddArray=myArray.filter(isNumeric);
document.writeln(oddArray);   // outputs: 1,3,5,7,9
```

This method can be prototyped to allow Internet Explorer and older browsers to use this method. Simply copy the following code into your Javascript toolbox and the `.filter()` method will be available regardless of your browser version.

```
//This prototype is provided by the Mozilla foundation and
//is distributed under the MIT license.
//http://www.ibiblio.org/pub/Linux/LICENSES/mit.license
if (!Array.prototype.filter)
{
  Array.prototype.filter = function(fun /*, thisp*/)
  {
    var len = this.length;
    if (typeof fun != "function")
      throw new TypeError();
    var res = new Array();
    var thisp = arguments[1];
    for (var i = 0; i < len; i++)
    {
      if (i in this)
      {
        var val = this[i]; // in case fun mutates this
        if (fun.call(thisp, val, i, this))
          res.push(val);
      }
    }
    return res;
  };
}
```

Supported Since: Firefox 1.5, Internet Explorer: ---

Array.forEach(function)

This is an odd little method. All it does is pass each element of the Array to the passed function. It ignores any results from the function and it returns nothing itself. It will pass all the Array contents through the function of your choice but the Array itself will not be affected and it will return nothing by itself.

This method will pass the current value, the current index, and a pointer to the array to your function -- `myfunction(curValue, curIndex, curArray)`.

```
var printArray = function (x, idx) {
   document.writeln('['+idx+'] = '+x);
}
var myArray = [1,'two',3,'four',5];
myArray.forEach(printArray); // outputs: [0] = 1 [1] = two [2] = 3 [3] = four [4] = 5
```

This method can be prototyped to allow Internet Explorer and older browsers to use this method. Simply copy the following code into your Javascript toolbox and the `.forEach()` method will be available regardless of your browser version.

```
//This prototype is provided by the Mozilla foundation and
//is distributed under the MIT license.
//http://www.ibiblio.org/pub/Linux/LICENSES/mit.license
if (!Array.prototype.forEach)
{
  Array.prototype.forEach = function(fun /*, thisp*/)
  {
    var len = this.length;
    if (typeof fun != "function")
      throw new TypeError();
    var thisp = arguments[1];
    for (var i = 0; i < len; i++)
    {
      if (i in this)
        fun.call(thisp, this[i], i, this);
    }
  };
}
```

Supported Since: Firefox 1.5, Internet Explorer: ---

Array.join(delimeter)

The join method will output your Array as a string with a delimiter of your choice. This is a very useful function if you need to convert an Array into something you can send back to the server. Simply pick a unique delimiter which is not likely to appear inside the data in your Array then you can turn it back into an Array on the server side.

```
var myArray = [1,'two',3,'four',5,'six',7,'eight',9,'ten'];
var test1 = myArray.join();
var test2 = myArray.join(', ');
var test3 = myArray.join('+');
document.writeln(test1+'<BR>'); // outputs: 1,two,3,four,5,six,7,eight,9,ten
document.writeln(test2+'<BR>'); // outputs: 1, two, 3, four, 5, six, 7, eight, 9,
ten
document.writeln(test3+'<BR>'); // outputs: 1+two+3+four+5+six+7+eight+9+ten
```

Supported Since: Netscape 3.0, Internet Explorer: 3.0

Array.indexOf(searchStr[, startIndex])

The `indexOf` method will search the array until it matches your search criteria. It will then return the index where the item was found. It will match only one item and the match must be exact. This is not as useful as the custom `.find()` method provided below in the useful prototypes section.

```
var myArray = [1,'two',3,'four',5,'six',7,'eight',9,'ten'];
document.writeln(myArray.indexOf('six'));      // outputs: 5
document.writeln(myArray.indexOf('not here')); // outputs: -1

To find and return all occurrences of an item you can use the following code…
var foundItems = [];
var index = array.indexOf(element)
while (index != -1)
{
  foundItems.push(index);
  index = array.indexOf(element, ++index);
}
```

This will create an array of indexes which match your search criteria (element) and store them in `foundItems[]`.

This method can be prototyped to allow Internet Explorer and older browsers to use this method. Simply copy the following code into your Javascript toolbox and the `.indexOf()` method will be available regardless of your browser version.

```
//This prototype is provided by the Mozilla foundation and
//is distributed under the MIT license.
//http://www.ibiblio.org/pub/Linux/LICENSES/mit.license
if (!Array.prototype.indexOf)
{
  Array.prototype.indexOf = function(elt /*, from*/)
  {
    var len = this.length;
    var from = Number(arguments[1]) || 0;
    from = (from < 0)
         ? Math.ceil(from)
         : Math.floor(from);
    if (from < 0)
      from += len;
    for (; from < len; from++)
    {
      if (from in this &&
          this[from] === elt)
        return from;
    }
    return -1;
  };
}
```

Supported Since: Firefox 1.5, Internet Explorer: ---

Array.lastIndexOf(searchStr[, startIndex])

`Array.indexOf()` searches from first to last, `lastIndexOf` searches from last to first.

```
var myArray = [1,'two',3,'four',5,'six',7,'eight',9,5,'ten'];
document.writeln(myArray.lastIndexOf(5));          // outputs: 9
document.writeln(myArray.lastIndexOf('not here')); // outputs: -1
```

This method can be prototyped to allow Internet Explorer and older browsers to use this method. Simply copy the following code into your Javascript toolbox and the `.lastIndexOf()` method will be available regardless of your browser version.

```
//This prototype is provided by the Mozilla foundation and
//is distributed under the MIT license.
//http://www.ibiblio.org/pub/Linux/LICENSES/mit.license
if (!Array.prototype.lastIndexOf)
{
  Array.prototype.lastIndexOf = function(elt /*, from*/)
  {
    var len = this.length;
    var from = Number(arguments[1]);
    if (isNaN(from))
    {
      from = len - 1;
    }
    else
    {
      from = (from < 0)
           ? Math.ceil(from)
           : Math.floor(from);
      if (from < 0)
        from += len;
      else if (from >= len)
        from = len - 1;
    }
    for (; from > -1; from--)
    {
      if (from in this &&
          this[from] === elt)
        return from;
    }
    return -1;
  };
}
```

Supported Since: Firefox 1.5, Internet Explorer: ---

Array.map(function)

The `map` method will call the provided function for each value of the array and it will return an array containing the results of those function calls.

The callback function is called with three arguments: the value, the index, and a pointer to the array being used respectively.

In the following example each element of the array is tested to see if it is numeric, if it is, it's passed into the new array, otherwise a zero is inserted.

```
var isNumeric = function(x) {
   // returns true if x is numeric and false if it is not.
   var RegExp = /^(-)?(\d*)(\.?)(\d*)$/;
   return String(x).match(RegExp);
}
var testElement = function(x) {
   if (isNumeric(x)) {
      return x;
   } else {
      return 0;
   }
}
var myArray = [1,'two',3,'four',5,'six',7,'eight',9,'ten'];
var newArray= myArray.map(testElement);
document.writeln(newArray); // outputs: 1,0,3,0,5,0,7,0,9,0
```

This method can be prototyped to allow Internet Explorer and older browsers to use this method. Simply copy the following code into your Javascript toolbox and the `.map()` method will be available regardless of your browser version.

```
//This prototype is provided by the Mozilla foundation and
//is distributed under the MIT license.
//http://www.ibiblio.org/pub/Linux/LICENSES/mit.license
if (!Array.prototype.map)
{
  Array.prototype.map = function(fun /*, thisp*/)
  {
    var len = this.length;
    if (typeof fun != "function")
      throw new TypeError();
    var res = new Array(len);
    var thisp = arguments[1];
    for (var i = 0; i < len; i++)
    {
      if (i in this)
        res[i] = fun.call(thisp, this[i], i, this);
    }
    return res;
  };
}
```

Supported Since: Firefox 1.5, Internet Explorer: ---

Array.pop()

The `pop()` method implements a basic stack structure in Javascript Arrays. `Array.pop()` will return the last element of an Array and delete it from the Array.

```
var myArray = [1,2,3,4,5,6,7,8,9,10];
document.writeln(myArray+'<BR>');          // outputs: 1,2,3,4,5,6,7,8,9,10
document.writeln(myArray.length+'<BR>');  // outputs: 10
var popped = myArray.pop();
document.writeln(popped+'<BR>');          // outputs: 10
document.writeln(myArray+'<BR>');         // outputs: 1,2,3,4,5,6,7,8,9
document.writeln(myArray.length+'<BR>'); // outputs: 9
```

Supported Since: Netscape 4.0, Internet Explorer: 5.5

Array.push(value[,value2[, value...]])

The `push()` method adds the passed value(s) to the end of the array. In addition to being incredibly useful for adding items to the array, it also allows the Array to emulate a basic stack structure.

```
var myArray = [1,2,3,4,5];
document.writeln(myArray+'<BR>');          // outputs: 1,2,3,4,5
document.writeln(myArray.length+'<BR>');  // outputs: 5
myArray.push(6);
document.writeln(myArray+'<BR>');          // outputs: 1,2,3,4,5,6
document.writeln(myArray.length+'<BR>');  // outputs: 6
myArray.push(7,8,9,10);
document.writeln(myArray+'<BR>');          // outputs: 1,2,3,4,5,6,7,8,9,10
document.writeln(myArray.length+'<BR>');  // outputs: 10
```

Supported Since: Netscape 4.0, Internet Explorer: 5.5

Array.reverse()

The `reverse()` method takes the array and reverses the order so the first item becomes the last and the last item becomes the first.

```
var myArray = [1,2,3,4,5,6,7,8,9,10];
document.writeln(myArray+'<BR>');          // outputs: 1,2,3,4,5,6,7,8,9,10
myArray.reverse();
document.writeln(myArray+'<BR>');          // outputs: 10,9,8,7,6,5,4,3,2,1
```

Supported Since: Netscape 3.0, Internet Explorer: 3.0

Array.shift()

The `shift()` method returns the first item in the Array, removing it from the Array in the process. Together with `Array.unshift`, `Shift` implements a basic queue structure in Javascript Arrays.

```
var myArray = [1,2,3,4,5,6,7,8,9,10];
document.writeln(myArray+'<BR>');          // outputs: 1,2,3,4,5,6,7,8,9,10
theItem = myArray.shift();
document.writeln(theItem+'<BR>');          // outputs: 1
document.writeln(myArray+'<BR>');          // outputs: 2,3,4,5,6,7,8,9,10
```

Supported Since: Netscape 5.5, Internet Explorer: 4.0

Array.slice([begin[, end]])

The `slice()` method copies a block of the array and returns it to your new variable. If you don't specify a beginning index the slice will begin at zero. If you don't specify and ending index the splice will continue to the end of the Array. So to make a copy of the array simply don't pass any arguments to the method.

```
var myArray = [1,2,3,4,5,6,7,8,9,10];
document.writeln(myArray+'<BR>');          // outputs: 1,2,3,4,5,6,7,8,9,10
var newArray = myArray.slice();
document.writeln(newArray+'<BR>');         // outputs: 1,2,3,4,5,6,7,8,9,10
var newArray = myArray.slice(5);
document.writeln(newArray+'<BR>');         // outputs: 6,7,8,9,10
var newArray = myArray.slice(5,7);
document.writeln(newArray+'<BR>');         // outputs: 6,7
```

Supported Since: Netscape 4.0, Internet Explorer: 4.0

Array.some(function)

The `some()` method will pass each element of the Array through the supplied function until true has been returned. If the function returns true some will in turn return true. If the entire array has been traversed and no true condition was found then `some()` will return false.

```
var isNumeric = function(x) {
   // returns true if x is numeric and false if it is not.
   var RegExp = /^(-)?(\d*)(\.?)(\d*)$/;
   return String(x).match(RegExp);
}
var myArray = ['one', 'two', 'three', 'four', 'five'];
document.writeln(myArray.some(isNumeric));   // outputs: false
var myArray = ['one', 'two', 3, 'four', 'five'];
document.writeln(myArray.some(isNumeric));   // outputs: true
```

This method can be prototyped to allow Internet Explorer and older browsers to use this method. Simply copy the following code into your Javascript toolbox and the `.some()` method will be available regardless of your browser version.

```
//This prototype is provided by the Mozilla foundation and
//is distributed under the MIT license.
//http://www.ibiblio.org/pub/Linux/LICENSES/mit.license
if (!Array.prototype.some)
{
  Array.prototype.some = function(fun /*, thisp*/)
  {
    var len = this.length;
    if (typeof fun != "function")
      throw new TypeError();
    var thisp = arguments[1];
    for (var i = 0; i < len; i++)
    {
      if (i in this &&
          fun.call(thisp, this[i], i, this))
        return true;
    }
    return false;
  };
}
```

Supported Since: Firefox 1.5, Internet Explorer: ---

Array.sort([function])

The `sort()` method, by default, will sort an Array alphabetically. If you would like to create your own sorting criteria, supply a function for the sort method to call. `Sort` will pass your function `(a,b)`. If a is less than b then return -1, if a is equal to b then return 0, if a is greater than b then return 1. Sort will take it from there.

```
var myArray=[8,10,50,5,7,83,24,19,168];
myArray.sort()
document.writeln(myArray);  // 10,168,19,24,5,50,7,8,83  (sorted alphabetically)
myArray.sort( function (a,b) { return a-b });  // Sort Numerically
document.writeln(myArray); //5,7,8,10,19,24,50,83,168
function compare(a, b) {
   // psudeo code.
   if (a < b) {
      return -1;
   }
   if (a > b) {
      return 1;
   }
   if (a == b) {
      return 0;
   }
}
myArray.sort(compare);
```

Supported Since: Netscape 3.0, Internet Explorer: 3.0

Array.splice(start[, howmany[, element1[,element...]]])

The `splice()` method at it's most basic allows you to delete an element from the array by simply specifying the index you'd like to delete and then how many elements you'd like to delete from that point. You can also specify any number of elements to insert into the array at that point.

`splice()` returns a new array containing the removed items. If you specify a starting number but don't specify how many, splice will truncate the array.

You can insert elements into the array without deleting anything by specifying zero for `howmany`.

```
var myArray=[1,2,3,4,5,6,7,8,9,10];
var newArray=[];

//delete one item at the 5th index.
newArray = myArray.splice(5,1);
document.writeln(myArray);  // outputs: 1,2,3,4,5,7,8,9,10
document.writeln(newArray); // outputs 6

//truncate the array at the 5th index.
myArray=[1,2,3,4,5,6,7,8,9,10];
newArray = myArray.splice(5);
document.writeln(myArray);  // outputs: 1,2,3,4,5
document.writeln(newArray); // outputs 6,7,8,9,10

// do nothing at all.
myArray=[1,2,3,4,5,6,7,8,9,10];
newArray = myArray.splice();
document.writeln(myArray);  // outputs: 1,2,3,4,5,6,7,8,9,10
document.writeln(newArray); // outputs undefined

// cut out the middle and insert 'blue', and 'green'
myArray=[1,2,3,4,5,6,7,8,9,10];
newArray = myArray.splice(1,8, 'blue', 'green');
document.writeln(myArray);  // outputs: 1,blue,green,10
document.writeln(newArray); // outputs 2,3,4,5,6,7,8,9

// Insert without deleting.
myArray=[1,2,3,4,5,6,7,8,9,10];
newArray = myArray.splice(5,0, '*');
newArray = myArray.splice(4,0, '*');
document.writeln(myArray);  // outputs: 1,2,3,4,*,5,*,6,7,8,9,10
```

Supported Since: Netscape 4.0, Internet Explorer: 5.5

Array.toSource()

The `toSource()` method is a Firefox only extension which takes the contents of an Array and returns the source code. That is, if you were to use the `toSource()` method and pass it through an `eval` statement you could rebuild the array. This is a useful debugging method and can be useful in Ajax/JSON applications. There is no prototype for this method, it works only in Firefox.

```
var myArray = ["a", "b", "c", {'sky':'blue', 'grass':'green'}];
var theSource = myArray.toSource()
document.writeln(theSource);    //   outputs:   ["a",   "b",   "c",   {sky:"blue",
grass:"green"}]
```

Supported Since: Firefox 1.5, Internet Explorer: ---

Array.toString()

This method outputs the contents of the Array as a string. It's not as powerful as `toSource()` since it doesn't expand objects but it is supported by browsers other than Firefox.

```
var myArray = ["a", "b", "c", {'sky':'blue', 'grass':'green'}];
var theSource = myArray.toString()
document.writeln(theSource);  // outputs: a,b,c,[object Object]
```

Supported Since: Netscape 3.0, Internet Explorer: 3.0

Array.unshift(value1[, value2[, value...]])

The `unshift` method inserts the value(s) passed in the method's arguments into the start of the Array. Together with the `shift(), unshift()` implements a basic queue structure in Javascript Arrays.

```
var myArray = [1,2,3,4,5,6,7,8,9,10];
document.writeln(myArray+'<BR>');         // outputs: 1,2,3,4,5,6,7,8,9,10
myArray.unshift('a','b','c');
document.writeln(myArray+'<BR>');         // outputs: a,b,c,1,2,3,4,5,6,7,8,9,10
```

Supported Since: Netscape 5.5, Internet Explorer: 4.0

Array.valueOf()

See `Array.toString()`.

Howto Delete An Element From An Array

You can delete an item from an Array with the `splice()` method. Simply supply the index of the item you wish to delete and how many items you want to delete and the item(s) will be removed for you.

```
var myArray=[1,2,3,4,5,6,7,8,9,10];
myArray.splice(5,1);        //delete one item at the 5th index.
document.writeln(myArray);  // outputs: 1,2,3,4,5,7,8,9,10
```

You if you would like to unset an element use the `delete` operator.

```
var myArray=[1,2,3,4,5];
delete myArray[3];
document.writeln(myArray); // outputs: 1,2,undefined,4,5
```

You can also use the `pop()` method to remove items from the end of the array and the `shift()` method to remove items from the beginning of the array.

Howto Clear-out/Reset An Array

Clearing out or resetting an array is as simple as assigning it a new empty bracket.

```
var myArray=[1,2,3,4,5,6,7,8,9,10];
document.writeln(myArray);  // outputs: 1,2,3,4,5,6,7,8,9,10
myArray = [];               // clear-out the array
document.writeln(myArray);  // outputs null
```

Alternatively you can set the length of the array to zero (`myArray.length=0`), likewise if you just need to clear out a few items at the end of the array, lower the length property by the number of items you would like removed from the end of the array.

Howto Tell The Difference Between An Array And An Object

Because Javascript's Array is just a modified Javascript object, it's actually not that easy to differentiate an Array and an Object, even when you absolutely need to. So here's a little function that will let you ask the array itself what it is. `isArray()` will return true if it's an array and false if it is not an Array.

```
function isArray(testObject) {
    return testObject && !(testObject.propertyIsEnumerable('length')) && typeof
testObject === 'object' && typeof testObject.length === 'number';
}
```

Usage…

```
var tmp = [5,9,12,18,'blue',30,7,97,53,33,30,35,27,30];
var tmp2 = {0:5,1:9,2:12}
test1 = isArray(tmp);    // returns true
test2 = isArray(tmp2);   // returns false;
```

Howto Easily Add Items To An Array

There are three ways to easily add items to an array. First you can use the `Array.length` property. Since Arrays start with index 0, then `Array.length` is always equal to the first empty index at the end of the Array.

```
var myArray = [];
myArray[0] = 'January';
myArray[1] = 'February';
document.writeln(myArray.length);          // Will output: 2
myArray[myArray.length] = 'March';         // Adds Item to end of Array
document.writeln(myArray.length);          // Will output: 3
document.writeln('0>'+myArray[0]+'<BR>');          // Will output: 0>January
document.writeln('1>'+myArray[1]+'<BR>');          // Will output: 1>February
document.writeln('2>'+myArray[2]+'<BR>');          // Will output: 2>March
```

You can add the `.push()` method of the Array. This will add the requested items to the end of the Array.

```
var myArray = [];
myArray[0] = 'January';
myArray[1] = 'February';
myArray.push('March');          // Adds Item to end of Array
document.writeln('0>'+myArray[0]+'<BR>');          // Will output: 0>January
document.writeln('1>'+myArray[1]+'<BR>');          // Will output: 1>February
document.writeln('2>'+myArray[2]+'<BR>');          // Will output: 2>March
```

You can use the `.unshift()` method to insert an item at the BEGINNING of the array!

```
var myArray = [];
myArray[0] = 'February';
myArray[1] = 'March';
myArray.unshift('January');         // Adds Item to beginning of Array
document.writeln('0>'+myArray[0]+'<BR>');          // Will output: 0>January
document.writeln('1>'+myArray[1]+'<BR>');          // Will output: 1>February
document.writeln('2>'+myArray[2]+'<BR>');          // Will output: 2>March
```

Howto Pass An Array Via Ajax

If you have an Array you need to package up and send via Ajax, you need to use the `join` method to turn your Array into a string. Find a unique character that is unlikely to appear inside your Array and use it as your delimiter. For most applications the tilde (~) character is a safe decimeter to use.

```
var myArray=[1,2,3,4,5,6,7,8,9,10];
var theData = myArray.join('~');     // theData=1~2~3~4~5~6~7~8~9~10
theData=encodeURIComponent(theData); // Optional but safe!  Escape the data
```

Now just send `theData` through your Ajax routine. On the server just do a `split('~')` in PhP to turn it back into an Array.

Howto Receive An Array Via Ajax

On the server side convert the Array into a string using a unique delimiter -- `implode` in PhP. (We're assuming a string with a tilde [~] delimiter in this example).

Once you've received the string from your Ajax handler simply use the string's `split` method to bust the string into an Array.

```
var ajaxStr = '1~2~3~4~5~6~7~8~9~10';
var myArray = ajaxStr.split('~');
document.writeln(myArray); // outputs: 1,2,3,4,5,6,7,8,9,10
```

Useful Prototypes

With prototypes you can extend the Array object and include any additional functionality you wish. For instance, the `Array.sort()` method sorts alphabetically by default but adding a numerical sort is as simple as including the following snippet of code in your toolbox.

```
Array.prototype.sortNum = function() {
   return this.sort( function (a,b) { return a-b; } );
}
```

All this does is make `sortNum` a method of Array just like `sort` and `splice` and `join` and all the other default methods. Any array can access it, even arrays that haven't been assigned to variables...

```
document.writeln([5,8,12,50,25,80,93].sortNum()); // outputs 5,8,12,25,50,80,93
```

Since the Javascript Array object is already so rich, there's not much left that needs to be done. However, there are a few snippets people have found helpful.

Useful Prototypes: Array.sortNum()

As stated above, just use the `sortNum()` method when you want to sort an array numerically instead of alphabetically.

```
Array.prototype.sortNum = function() {
   return this.sort( function (a,b) { return a-b; } );
}
```

Useful Prototypes: Array.find(searchStr)

`Array.indexOf()` is a nice method but this extension is a little more powerful and flexible. First it will return an array of all the indexes it found (it will return false if it doesn't find anything). Second in addition to passing the usual string or number to look for you can actually pass a regular expression, which makes this the ultimate Array prototype in my book.

```
Array.prototype.find = function(searchStr) {
  var returnArray = false;
  for (i=0; i<this.length; i++) {
    if (typeof(searchStr) == 'function') {
      if (searchStr.test(this[i])) {
        if (!returnArray) { returnArray = [] }
        returnArray.push(i);
      }
    } else {
      if (this[i]===searchStr) {
        if (!returnArray) { returnArray = [] }
        returnArray.push(i);
      }
    }
  }
  return returnArray;
}
```

Usage…

```
var tmp = [5,9,12,18,56,1,10,42,'blue',30, 7,97,53,33,30,35,27,30,'35','Ball', 'bubble'];
//         0/1/2 /3 /4/5 /6 /7     /8  /9/10/11/12/13/14/15/16/17/  18/    19/      20
var thirty=tmp.find(30);              // Returns 9, 14, 17
var thirtyfive=tmp.find('35');        // Returns 18
var thirtyfive=tmp.find(35);          // Returns 15
var haveBlue=tmp.find('blue');        // Returns 8
var notFound=tmp.find('not there!'); // Returns false
var regexp1=tmp.find(/^b/);           // returns 8,20    (first letter starts with b)
var regexp1=tmp.find(/^b/i);          // returns 8,19,20 (same as above but ignore case)
```

Useful Prototypes: Array.shuffle()

The anti-sort, this `shuffle()` method will take the contents of the array and randomize them. This method is surprisingly useful and not just for shuffling an array of virtual cards.

```
Array.prototype.shuffle = function (){
   for(var rnd, tmp, i=this.length; i; rnd=parseInt(Math.random()*i), tmp=this[--
i], this[i]=this[rnd], this[rnd]=tmp);
};
```

Usage…

```
var myArray = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15];
myArray.shuffle();
document.writeln(myArray); // outputs~: 8,1,13,11,2,3,4,12,6,14,5,7,10,15,9
```

Useful Prototypes: Array.compare(array)

If you need to be able to compare Arrays this is the prototype to do it. Pass an Array you want to compare and if they are identical the method will return true. If there's a difference it will return false. The match must be identical so '80' is not the same as 80.

```
Array.prototype.compare = function(testArr) {
    if (this.length != testArr.length) return false;
    for (var i = 0; i < testArr.length; i++) {
        if (this[i].compare) {
            if (!this[i].compare(testArr[i])) return false;
        }
        if (this[i] !== testArr[i]) return false;
    }
    return true;
}
```

Usage...

```
var myArray = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15];
var yourArray = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15];
document.writeln(myArray.compare(yourArray)); // outputs: true;
yourArray[0]='1';
document.writeln(myArray.compare(yourArray)); // outputs: false;
yourArray[0]='one';
document.writeln(myArray.compare(yourArray)); // outputs: false;
yourArray[0]=1;
document.writeln(myArray.compare(yourArray)); // outputs: true;
```

Version

This document is current as of Firefox 2.03, IE 7.0, JSCRIPT 5.6, ECMA-262 Edition 3, Javascript 1.7

License