Arrays in JavaScript

Lecture 6
Chapter 7 in JavaScript text

Arrays in JavaScript

We’ve seen how objects in JavaScript are very different from objects in other OO languages you’ ve seen.

Now we take a look at arrays - which also have some very important differences.

Arrays are special objects

An object is an **unordered** collection of **named** properties.

An array is an **ordered** collection of elements.

Each element has a **position** indexed by a **numeric** value.

Array characteristics

- As usual, array indexes start at 0.

- Array elements are **untyped**
  - You may put any type of data in each element - they do not need to be all of the same type!

- Arrays can be **sparse**.

Creating Arrays

Like objects, arrays have a **literal** notation

```
var myArray = [“hello”, “world”, 1, 2, 3];
var myMultiArray = [ [ 1, 2, 3], [4, 5, 6] ];
var myObjectArray = [
    { first: “George”, last : “Washington” },
    { first : “Abe”, last : “Lincoln” }
    ];
```

More JSON: An object can have an array as a property.

```
var obj = { a : [1,2,3], b : [‘a’,‘b’,‘c’], c : [x,y,z] };
```

Creating Arrays

You can create an empty array using literal notation

```
var empty = [ ];
```

An array can also be created with a constructor

```
var empty = new Array();
var a = new Array(10); // size of 10.
var b = new Array(1, 2, 3, 4); // int with 4 elements
```

Reading / Writing

Array indexes work a lot like object properties and variables - they are defined “on write”.

```
var a = [ “world” ];
console.log( a[0] );
a[1] = 3.14;      // creates index 1
a[2] = “hello”; // creates index 2
console.log (a[2] + “ “ + a[0]);
console.log (a.length);
```

Indexes and Properties

- Arrays **<u>are</u>** objects. They have properties.
- Indexes (integers, 0-2^{32}) are special property names
  - They are automatically converted to strings
  - They cause the length property to be maintained
- However - you can add a property to an array...

```
var a = [1, 2, 3];
a["3"] = 4;  // converts to a[3];
a["Four"] = 5;  // creates a normal property
console.log(a.Four);
console.log(a.length);
for ( i = 0; i < a.length; i++ ) {
    console.log (i + " -> " + a[i]);
}
```

Sparse arrays

Unlike C++ arrays, elements aren't **necessarily** stored in contiguous memory.

```
var a = [1, 2, 3];
a[5] = 512;
console.log(a[3]);  // prints undefined
console.log(a.length);
```

`length` returns an index larger than the largest integer…

```
for ( i = 0; i < a.length; i++ ) {
   console.log (i + " -> " + a[i]);
}
```

Length is not only for reading...

Oddly, the length property is **writable**.

```
var a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
console.log(a[5]);
console.log(a.length);
a.length = 3;
for ( i = 0; i < a.length; i++ ) {
    console.log (i + " -> " + a[i]);
}
console.log(a[5]);
```

Arrays as Stacks

Arrays are very versatile, and have numerous methods defined on them.

```
var stack = [];
stack.push(1);
stack.push(2, 3);
for ( i = 0; i < stack.length; i++ ) {
    console.log (i + " -> " + stack[i]);
}
stack.pop();
for ( i = 0; i < stack.length; i++ ) {
    console.log (i + " -> " + stack[i]);
}
```

Deleting elements

Elements can be removed, but note this can create *sparse* arrays (elements are not shifted)

```
var a = [1, 2, 3];
delete a[1];
for ( i = 0; i < a.length; i++ ) {
    console.log (i + " -> " + a[i]);
}
for ( i = 0; i < a.length; i++ ) {
    if ( a[i] === undefined ) continue;
    console.log (i + " -> " + a[i]);
}
```

Now a[1] will be **undefined**

More methods

join - converts all elements of array to string form

```
var a = [1, 2, 3];
console.log(a.join()); // default to comma separators
console.log(a.join(“+”));
```

reverse - does what the name implies

```
a.reverse();
console.log(a.join());
```

Sorting

```
var a = [“banana”, “cherry”, “apple” ];
a.sort();
console.log(a.join());
```

You may supply a function to sort, which is used when comparing… we’ll see this in the next segment

Search with `indexOf`

```
var a = [1, 2, 3, 4, 1, 2, 1, 2];
console.log( a.indexOf(1) );
console.log( a.lastIndexOf(1));
console.log(a.indexOf(5));
console.log(a.indexOf(1, 1));
console.log( a.lastIndexOf(1, 5));
```

- indexOf takes a second argument, which indicates the start position to search from (searching right)
- lastIndexOf takes an argument for the position to start its search (searching left)
- If a value isn’t found, -1 is returned.

Array operations

concat can add arrays together

```
var a = [0, 1, 2];
var b = [3, 4, 5];
var c = a.concat(b);
console.log(c.join());
```

slice returns a **new** sub-array

```
console.log(c.slice(0, 3));
console.log(c.slice(2, 4));
console.log(c.slice(3, -1));
```

Next...

Next we look at the last fundamental piece of JavaScript we need to learn before really starting to do server-side development with Node

Functions!