App Development & Modelling

BSc in Applied Computing

Produced by

Eamonn de Leastar (edeleastar@wit.ie)

Department of Computing, Maths & Physics
Waterford Institute of Technology

http://www.wit.ie

http://elearning.wit.ie

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

eLearning support unit

Arrays

Creating & Using Arrays

- Arrays are a type of object that are ordered by the index of each item it contains.

- The index starts at zero and extends to however many items have been added, which is a property of the array known as the "length" of the array.

- Similar to objects, an array can be created with the array constructor or the shorthand syntax known as array literal.

Creating Arrays...

Array Constructor

```
var foo = new Array;
```

Array Literal

```
var bar = [];
```

Creating Arrays with Dimensions

Array Constructor

```
var foo = new Array(100);
```

Array Literal

```
var bar = [];
```

Using Arrays...

- Insertion into arrays can be via:
  - [] notation (like Java)
  - Using ‘push’ which appends to the end of the array

```
var foo = [];

foo.push("a");
foo.push("b");

alert( foo[ 0 ] ); // => a
alert( foo[ 1 ] ); // => b

alert( foo.length ); // => 2

foo.pop();

alert( foo[ 0 ] ); // => a
alert( foo[ 1 ] ); // => undefined

alert( foo.length ); // => 1
```

Definition of an Array

- Arrays are zero-indexed, ordered lists of values.

- They are a handy way to store a set of related items of the same type (such as strings), though in reality, an array can include multiple types of items, including other arrays.

- To create an array, either use the object constructor or the literal declaration, by assigning the variable a list of values after the declaration

```
// A simple array with constructor
var myArray1 = new Array( "hello", "world" );
// literal declaration, the preferred way
var myArray2 = [ "hello", "world" ];
```

- The literal declaration is generally preferred. See the Google Coding Guidelines for more information.

Push

- If the values are unknown, it is also possible to declare an empty Array, and add elements either through functions or through accessing by index:

- 'push' is a function that adds an element on the end of the array and expands the array respectively. You also can directly add items by index. Missing indices will be filled with 'undefined'.

```
// Creating empty arrays and adding
values
var myArray = [];

// adds "hello" on index 0
myArray.push("hello");

// adds "world" on index 1
myArray.push("world");

// adds "!" on index 2
myArray[ 2 ] = "!";
```

Missing Indices

- Missing indices will be filled with 'undefined'.

- If the size of the array is unknown, 'push' is far more safe.

```
// Leaving indices
var myArray = [];

myArray[ 0 ] = "hello";
myArray[ 1 ] = "world";
myArray[ 3 ] = "!";

console.log( myArray );
// [ "hello", "world", undefined, "!" ];
```

- You can both access and assign values to array items with the index

```
// Accessing array items by index
var myArray = [ "hello", "world", "!" ];

console.log( myArray[2] ); // "!"
```

Array Methods and Properties

length

- The .length property is used to determine the amount of items in an array.

```
// Length of an array
var myArray = [ "hello", "world", "!" ];

console.log( myArray.length ); // 3
```

- You will need the .length property for looping through an array:

```
// For loops and arrays - a classic
var myArray = [ "hello", "world", "!" ];

for ( var i = 0; i < myArray.length; i = i + 1 )
{
  console.log( myArray[i] );
}
```

- Except when using for/in loops:

```
// For loops and arrays - alternate method
var myArray = [ "hello", "world", "!" ];

for ( var i in myArray )
{
  console.log( myArray[ i ] );
}
```

concat

- Concatenate two or more arrays with .concat:

```
// Concatenating Arrays
var myArray = [ 2, 3, 4 ];
var myOtherArray = [ 5, 6, 7 ];

// [ 2, 3, 4, 5, 6, 7 ]
var wholeArray = myArray.concat( myOtherArray );
```

join

- .join creates a string representation of the array. Its parameter is a string that works as a separator between elements (default separator is a comma):

```
// Joining elements
var myArray = [ "hello", "world", "!" ];

console.log( myArray.join(" ") );  // "hello world !";
console.log( myArray.join() );     // "hello,world,!"
console.log( myArray.join("") );   // "helloworld!"
console.log( myArray.join("!!") ); // "hello!!world!!!";
```

pop

- .pop removes the last element of an array. It is the opposite method of .push:

```
// pushing and popping
var myArray = [];

myArray.push( 0 ); // [ 0 ]
myArray.push( 2 ); // [ 0 , 2 ]
myArray.push( 7 ); // [ 0 , 2 , 7 ]
myArray.pop();     // [ 0 , 2 ]
```

reverse

- The elements of the array are in reverse order after calling this method:

```
// reverse
var myArray = [ "world" , "hello" ];

// [ "hello", "world" ]
myArray.reverse();
```

shift

- Removes the first element of an array.

- With .pop and .shift, you can simulate a queue:

```
// queue with shift() and pop()
var myArray = [];

myArray.push( 0 ); // [ 0 ]
myArray.push( 2 ); // [ 0 , 2 ]
myArray.push( 7 ); // [ 0 , 2 , 7 ]
myArray.shift();   // [ 2 , 7 ]
```

forEach

- In modern browsers it is possible to traverse through arrays with a .forEach method, where you pass a function that is called for each element in the array.
- The function takes up to three arguments:
  - Element - The element itself.
  - Index - The index of this element in the array.
  - Array - The array itself.
- All of these are optional, but you will need at least the 'element' parameter in most cases

```
// native forEach
function printElement( elem )
{
  console.log( elem );
}


myArray = [ 1, 2, 3, 4, 5 ];

// prints all elements to the console
myArray.forEach( printElement );
```

Element & Index Parameters

```
function printElementAndIndex( elem, index )
{
  console.log( "Index " + index + ": " + elem );
}


// prints "Index 0: 1" "Index 1: 2" "Index 2: 3" ...
myArray.forEach( printElementAndIndex );
```

Exercise 5.4

- In your current project (called js-lab-2) create a file called 'arrays.js' in the js folder. Create a HTML file to load this script.
  - In arrays.js do the following:
  - Declare an array called 'todolist'.
  - Initialise this array to contain 5 strings, which describe some simple tasks (make up some).
  - write a for loop then logs each of these tasks to the console
- Use google chrome developers tools -> Console to verify that it works as expected. Implement one more experiment:
  - pass the entire array to a single call to console.log
- Observe the result.

Exercise 5.5

- Take the last code fragment - foreach above, and incorporate into your arrays.js file.

- In Chrome, load the html page and single step through each of the methods.