Working with Arrays

# **7-1 The difference between and array instance and an array literal**

## **Problem**

What is the difference between arrays that are created using [] or new Array() .

## **Solution**

An array literal ([]) is a quick way to make an array. The major difference is when creating an instance using the new operator you can pass a parameter to the constructor that will either give the array a length or values.

## **The Code**

Listing 7-1. Creating both array literals and instances of an array object.

var arrayLit = ['insoc', 'DM', 'Erasure'];  
var arrayObj = new Array(3);

console.log(arrayLit.length); //returns 3  
console.log(arrayObj.length); //returns 3

console.log(arrayLit[0]); //returns insoc  
console.log(arrayObj[0]); // returns undefined  
  
var arrayList = new Array(1,2,3);  
  
console.log(arrayList.length); //returns 3  
console.log(arrayList[0]); //returns 1

## **How it Works**

In both instances you get an array with all the properties and methods of an array. Using the array literal is a quick way of building and array and is considered a very easy to read way of generating an array. Using the array constructor, depending on the amount and type of parameters, you ether set the length of the array or add values to it. For example new Array(3) will give you a length of 3 with all the values being undefined. Using literals would prevent confusion.

# **7-2 How do you create a multi-dimensional array using literal notation**

## **Problem**

You need to create an array inside an array.

## **Solution**

When creating the array, add another array inside an element.

## **The Code**

Listing 6-2. Getting date information then time information

var now = new Date(); //returns todays date and time

console.log(now.getDate()); //returns the day of the month from 1 to 31   
 console.log(now.getDay()); //returns the day of the week its zero based like an array 0 - 6;  
 console.log(now.getFullYear()); //returns the current year  
 console.log(now.getHours()); //returns hours from 0-23  
 console.log(now.getMonth()); //return month from 0-11

//time information

console.log(now.getSeconds()); //returns seconds from 0-59  
 console.log(now.getTime()); //returns the amount of milliseconds since the first of January 1970

## **How it Works**

There are a lot of methods that will give you most of the date or time information you need. In some cases, the values are treated similar to an array, being zero based. For example running the getMonth() if its January the result will be 0.

These methods can be put together to add give your users things like the current date and time. Now we can do interesting things with this information.

# **7-3 Use the Map method to reverse a string**

## **Problem**

You need to reverse the contents of an array without writing a for loop.

## **Solution**

The map method will call a function on every element of the array.

## **The Code**

Listing 7-3. Using the map method to reverse a string and remove numbers

//take a string convert it into an array and reverse the order

var startString = 'racecar';

var resultString = startString.split('').map(function(e){

return e;

}).reverse().join('');

console.log(resultString); //result is racecar. It's a palendrome!

## **How it Works**

In this example we split the string. JavaScript will turn it into an array. This gives us access to the map() method where we can perform work on each element of the array. In this case we just return the value where it gets reversed and joined back into a string.

# **7-4 Using the Map method to remove duplicate items from an array**

## **Problem**

You need to remove duplicate items from an array.

## **Solution**

Here we can also use the map() method. We check each item in the array and use a function to figure out if each item is unique.

## **The Code**

Listing 7-4. Removing duplicate items from an array.

//remove duplicate items from an array;

var itemExists = {};

var numberArray = [1,2,2,3,4,5,6,6,6,6,6,7,8,9];

var fixedArray = [];

numberArray.map(function(e){

if(!itemExists[e]){

itemExists[e] = e;

fixedArray.push(e);

}

});

console.log(fixedArray); // result is [1,2,3,4,5,6,7,8,9];

## **How it Works**

Running the map() method you assign a function that lets you evaluate each element. Here you perform an if statement where you check if the itemExists object has a property based on the value of the element. If the property does not exist, then create the property and add the current element to fixedArray. If it does then ignore everything and go to the next element.

# **7-5 Merge the value of two different arrays**

## **Problem**

You need to make a new array based on the values of two arrays.

## **Solution**

Use the concat method on the first array and pass the second array as it’s value.

## **The Code**

Listing 7-5. Using the concat method to merge two arrays

//merting two arrays  
  
var array1 = [1,2,3];  
var array2 = [4,5,6];  
var combinedArray = array1.concat(array2);

## console.log(combinedArray); //returns 1,2,3,4,5,6

## **How it Works**

Using the concat() method will merge the values of an array to the end of the array that is calling the method. This method can add multiple arrays by separating by separating them by commas. The result is a new array that copies references from the original array.

# **6-6 Make sure every element in an array meets certain criteria**

## **Problem**

You want to make sure that every element in the array is greater than zero.

## **Solution**

The every() method will call a function on every element in the array, here you can check the value of each element.

## **The Code**

Listing 7-6. Checking if every element in the array was larger than zero

// check if each item is larger than zero

var biggerThanZero = [1,2,3,4,5,6,7,8,9].every(function(element){

return element > 0;

});

console.log(biggerThanZero); //returns true

}

## **How it Works**

This will happen until the function returns a falsy result (a false value when converted to a Boolean). If none of the items. If such a result does not occur, then the function returns true. If any element in the array had a value of zero, then it would return false. Some of the other properties that can be used is index, and the array itself.

# **6-7 Formatting date and time.**

## **Problem**

You want to format dates, including language and time zone.

## **Solution**

The date can be formatted not only by using methods like the getDate() method. It can also be formatted to use different languages. Using either toLocaleDateString, toLocaleDateSting or the internationalization API. The locales, options and internationalization API works with the latest desktop browsers and safari’s nightly build. Mobile support is only for Chrome for Android 26.

## **The Code**

Listing 6-7. Using toLocaleDateString and ECMAScript Internationalization API

//format the date using toLocaleTimeString

console.log(currentDate.toLocaleDateString('ja-JP'));

console.log(currentDate.toLocaleDateString('ja-JP', {weekday: 'long', year:'numeric', month:'short', day:'numeric', hour:'2-digit', minute:'2-digit'}))

new Intl.DateTimeFormat('ja-JP', { weekday: 'long', year:'numeric', month:'short', day:'numeric', hour:'2-digit', minute:'2-digit', timeZone:'Asia/Tokyo', timeZoneName: 'short'}).format(new Date(1975, 07, 19)); //returns 1975年8月19日火曜日 13:00 JST

new Intl.DateTimeFormat('en-US', { weekday: 'long', year: 'numeric', month: 'long', day: 'numeric' }).format(new Date(1975, 07, 19)); // returns Tuesday, August 19, 1975

## **How it Works**

Using toLocaleDateString or toLocaleTimeString the first parameter is the locales that can be used. Some examples would be ’en-GB’ for British English or ‘ko-KR’ for Korean.

The second parameter outlines the format details. This parameter is optional. Here you can set things like the time zone that you want to use. The times zones must recognize UTC, it uses the runtime as the default. It also understands names from the IANA time zone database like ‘Asia/Tokyo’ or ‘America/New\_York’.

Other options include: weekday, era, year, month, day, hour minute, second and timeZoneName.

Using the internationalization API you can format the date the same way as using the toLocale methods. In the constructor, you can set the local, as the first parameter. The second parameter uses the same options object. After that you can use the format method and pass the date object.

# **6-8 Calculate days from a certain date**

## **Problem**

You need to figure out how much time has passed since a certain date.

## **Solution**

Use the getTime() method and check if your date is greater than the start start date and less than the end date.

## **The Code**

Listing 6-8. Calculating the how many days from a certain date.

var date1 = new Date('6/15/2016);

var date2 = new Date('6/19/2016');

// 86,400,000 milliseconds is the number of milliseconds in one day  
var oneDayInMS = 1000 \* 60 \* 60 \* 24;

// Calculate the difference in milliseconds  
var differenceInMS = Math.abs(date1.getTime() - date2.getTime());  
var days = Math.round(differenceInMS/oneDayInMS);

console.log(days);

## **How it Works**

You create two date objects like in similar examples. In this case we also subtract the first date from the second by milliseconds. The result on it’s own would be a negative new number, this is why the equation is done inside the Math.abs() method witch will turn our number into a positive integer.

We also have a variable that is the amount of milliseconds of a day. It is expressed by 1000 milliseconds times 60 seconds, times 60 minutes, times 24 hours. This will give you 86,400,000 milliseconds. Divide the difference between the two days by the amount of milliseconds in a whole day and round to the next highest integer.

.

# **6-9 Check if your date is in a range of dates**

## **Problem**

You need to know if the date selected is within a specific range.

## **Solution**

The abs method of the Math objet will give you the non-negative number.

## **The Code**

Listing 6-9. Check if the selected date is within the range of the start date and the end date.

var startDate = new Date('2/5/1990');  
var endDate = new Date('3/25/2013');  
  
var selectedDate = new Date('12/3/2000');  
  
if(selectedDate.getTime() > startDate.getTime() && selectedDate.getTime() < endDate.getTime()){  
 console.log('In the range');  
 }else{  
 console.log('Not in the range');   
}

## **How it Works**

The getTime() method returns time in milliseconds UTC time (number of milliseconds since January 1st 1970). In this case we are just testing If the date chosen is higher than the start time in milliseconds and lower than the end time in milliseconds. Both parts of the IF statement need to be true for this example to work.