



# JavaScript and jQuery Course

## Instructor Teresa Pelkie

### Session 6

## Topic: How to use Arrays

Chapter 5: How to work with arrays — page 141 to end

### Arrays

An *Array* is an object or a special type of variables that is used to store multiple values using a single variable name. Arrays are used to store values that are represented as a list or collection. Arrays are supported in JavaScript using the `Array()` object - uppercase A.

The maximum length, or maximum number of elements allowed in an array is 4,294,967,295.

### How to create an array

There are three commonly used techniques to define an array:

- Define the array as a separate var using the JavaScript `new` keyword. You then assign values to the array elements in subsequent statements.
- Define the array using the JavaScript `new` keyword and initialize it within the same statement.
- Define the array without the `new` keyword and initialize it within the same statement.

The **new** keyword invokes the `Array` object *constructor*, which creates a new instance of the `Array` object, using the name that you specify.

### Referring to elements within the array

Array elements are referenced by the **index position** of the element that you need to access. The index is zero-based, meaning that the first element in the array is referred to as "element 0 (zero)".

To access an array element at a specific position in the array, you use the name of the array followed by the index value enclosed in square brackets (`[ ]`).

**Example 1:** Define an array, assign four array element values in consecutive positions, beginning with the first element in the array.

```
var myArray = new Array();  
  
myArray[0] = "apples";  
myArray[1] = "oranges";  
myArray[2] = "pears";  
myArray[3] = "peaches";
```

In this example, an array named `myArray` is defined as a new `Array` object. The opening and closing parentheses characters are required after the `Array` object name. In this example, since there are no parameters being passed to the `Array` constructor, there are no values between the parentheses.

You can use this type of definition and initialization when you do not know what the values are that are to be assigned to the array elements until later in your code. For example, you might need to get input from the user on web form before you know what the values are.

**Example 2:** Define an array and assign four array element values in the same statement, using the new keyword.

```
var myArray = new Array("apples", "oranges", "pears", "peaches");
```

In this example, an array named `myArray` is defined as a new `Array` object. Because the values that are to be used in the array are known when the array is defined, the values can be passed as parameters to the constructor by putting the values inside the parentheses.

**Example 3:** Define an array and assign four array element values in the same statement.

```
var myArray = ["apples", "oranges", "pears", "peaches"];
```

In this example, an array named `myArray` is defined and initialized with four values. The new keyword is assumed and the constructor is implicitly invoked (meaning that you did not specify `new Array`). JavaScript provides this "shorthand" way of defining and initializing an array because it is very common to have an array definition immediately followed by initializations for the array elements.

When you use this type of array definition, you use square bracket characters `[` and `]` to enclose the list of array elements.

**Example 4:** Define an array with no initializations in the definition, using the "shorthand" initialization.

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

In this example, an array named `myArray` is defined. There are no initialization values supplied, so the empty list is used.

This example is similar to Example 1 above, where the array definition was specified on one statement, with array element initialization statements on other statements.

## Some facts concerning arrays

- Arrays are zero-based, meaning that you access elements beginning at **0**
- Elements in the array are accessed via their **index value**, which is the element number, enclosed in square brackets
- An array has a **length property**, which returns the count of the number of elements in the array

**Example 5:** Retrieve the length of an array

```
var myArray = new Array();  
myArray[0] = "apples";  
myArray[1] = "oranges";  
myArray[2] = "pears";  
myArray[3] = "peaches";  
alert("The length of myArray is " + myArray.length);
```

The length of myArray is 4

In this example, the `length` property of the array is used to retrieve the current number of elements in the array.

**Example 6:** Retrieve the value a single array element

```
var myArray = new Array();  
myArray[0] = "apples";  
myArray[1] = "oranges";  
myArray[2] = "pears";  
myArray[3] = "peaches";  
alert("myArray[0] is " + myArray[0]);
```

In this example, the value of the first element in the array is retrieved with the `myArray[0]` array element reference.

**Example 7:** Retrieve the values of all the elements in the array

```
var myArray = new Array();  
myArray[0] = "apples";  
myArray[1] = "oranges";  
myArray[2] = "pears";  
myArray[3] = "peaches";  
alert(myArray);
```

apples,oranges,pears,peaches

In this example, all of the elements in the array are accessed simply by referring to the array name. When the array is accessed this way, it is represented as a comma-delimited list of value.

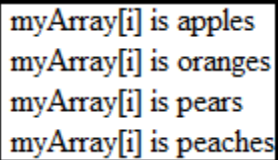
**Example 8:** Looping through an array

```
var myArray = new Array();

myArray[0] = "apples";
myArray[1] = "oranges";
myArray[2] = "pears";
myArray[3] = "peaches";

for (var i = 0; i < myArray.length; i++)
{
    document.write("myArray[i] is " + myArray[i] + "<br>");
}
```

The output, using document.write() for convenience only



```
myArray[i] is apples
myArray[i] is oranges
myArray[i] is pears
myArray[i] is peaches
```

Because arrays frequently contain elements located in sequential positions, it is very common to use a for loop like what is shown in this example to access the elements in sequence. Note that the loop counter (i) is initialized to zero, since the array elements are accessed starting at element zero. The length property is used to determine how many times the loop is to be processed. After each iteration, the value of i is incremented by 1, giving access to the next element in the array on the next pass through the loop.

**Example 9:** Looping through an array using a for-in loop

```
var myArray = new Array();

myArray[0] = "apples";
myArray[1] = "oranges";
myArray[2] = "pears";
myArray[3] = "peaches";

for (var i in myArray)
{
    alert("myArray[i] is " +
        myArray[i] +
        "and i is " + i);
}
```

The for-in loop is a special type of looping construct that can only be used with arrays. This construct combines the initialization, testing, and incrementing of the index variable (in this example, the variable named i) into a compact format that can be used to access the elements of the underlying array.

### Example 10: Modify the value an array element

```
var myArray = new Array();

myArray[0] = "apples";
myArray[1] = "oranges";
myArray[2] = "pears";
myArray[3] = "peaches";

myArray[2] = "red pears";

alert("myArray[2] is " + myArray[2]);
```

In this example, the third element in the array is set to a different value.

### Array Methods (some are not in the text)

#### join()

The `join()` method joins the elements of an array into a string, and returns the string. The elements are separated by an optional separator character or characters. The default separator is a comma (,).

```
var myArray = new Array("apples", "oranges", "pears", "peaches");
var energy = myArray.join();
```

The value of `energy` is the string `apples,oranges,pears,peaches`

#### concat()

The `concat()` method joins two or more arrays. This method does not change the existing arrays, but returns a new array, containing the values of the joined arrays.

```
var newArray = myArray.concat(thisArray, thatArray);
```

#### indexOf()

The `indexOf()` method searches the array for the specified item and returns its position.

The search starts at the specified position, or at the beginning if no start position is specified. The search is ended at the end of the array. If the item is not found, the value returned is `-1`. If the item is occurs more than once in the array, the value returned is the position of the first occurrence.

```
var myArray = new Array("apples", "oranges", "pears", "peaches");
var myIndex = myArray.indexOf("apple");

document.write(myIndex); // value is 0 (location of the element in the array)
```

#### pop()

The `pop()` method removes the last element of an array and returns the value of the element.

```
var myArray = new Array("apples", "oranges", "pears", "peaches");
var lastElement = myArray.pop();

document.write(lastElement ); // the value is peaches
```

## shift()

The `shift()` method removes the first element of an array and returns the value of the element.

```
var myArray = new Array("apples", "oranges", "pears", "peaches");
var firstElement = myArray.shift();

document.write(firstElement );    // value is apples
```

## push()

The `push()` method adds new items to the end of an array and returns the new length.

```
var myArray = new Array("apples", "oranges", "pears", "peaches");
var newLastElement = myArray.push("mango", "cherries");

document.write(newLastElement );    // value of newLastElement is 6
```

## unshift()

The `unshift()` method adds new items to the beginning of an array and returns the new length.

```
var myArray = new Array("apples", "oranges", "pears", "peaches");
var newFirstElement = myArray.unshift("kiwi", "melon");

document.write(newFirstElement );    // value of newFirstElement is 6
```

## sort()

The `sort()` method sorts the items of an array. Default sort order is alphabetical, ascending.

## reverse()

The `reverse()` method reverses the order of the elements in an array

## slice()

The `slice()` method returns the selected elements in an array as a new array object. The `slice()` method selects the elements starting at the given start argument, and ends at, but does not include, the given end argument. The original array will not be changed.

```
var myArray = new Array("apples", "oranges", "pears", "peaches");
var newArray2 = myArray.slice(1, 3);

document.write(newArray2);    // newArray2 is "oranges","pears"
```

## splice()

The `splice()` method adds/removes items to/from an array, and returns the removed item(s). This method changes the original array. The parameters are:

- `index` — Index at which to start changing the array.
- `howMany` — An integer indicating the number of old array elements to remove. If `howMany` is 0, no elements are removed.
- `element1, ..., elementN` — The elements to add to the array. If no elements are specified, `splice()` simply removes elements from the array.

```
var myNewArray3 = myArray3.splice(2,2,"kiwi");
```

## Sample Files for this session

| File Name                          | Description  |
|------------------------------------|--|
| 04_date_other_ways_to_display.html | date object, displayed in different formats.   |
| 04_get_day_name_array.html         | Get the day name, using an array of day names.   |
| 04_get_day_name_if_condition.html  | Get the day name, using an if/else if sequence.  |
| 06_array_delete.html               | This page shows how to delete an element from an array using the delete command.   |
| 06_array_join.html                 | This page shows how to use the join method of the Array object.  |
| 06_array_loop_for-in.html          | This page shows how to use the for-in loop construct with an array.  |
| 06_array_loop_for.html             | This page shows how to use a for loop to loop through an array   |
| 06_array_loop_for_2.html           | This page shows how to use a for loop to loop through an array and build a string of the array elements.                   |
| 06_array_modify.html               | This page shows how to modify an array element.  |
| 06_array_pop.html                  | This page shows how to use the pop method of an Array object to remove the last element of the array.                      |
| 06_array_push.html                 | This page shows how to use the push method of an Array object to add a new element at the end of the array.                |
| 06_array_shift.html                | This page shows how to use the shift method of an Array object to remove the first element of the array.                   |
| 06_array_simple.html               | This page shows how to create and initialize an array.   |
| 06_array_simple2.html              | This page shows how to create and initialize an array, example 2.  |
| 06_array_simple3.html              | This page shows how to create and initialize an array, example 3.  |
| 06_array_unshift.html              | This page shows how to use the unshift method of an Array object to add a new element before the first element of an array |
| 06_textbook_email_list.html        | This file contains the email list example that is presented in the textbook.   |