Course Outline

Lecture 4: Array and Math Objects' Methods

4-1 Array Object

4-2 Additional Array Methods

4-3 Math Object

Arrays Objects

For example, an array called **myArray** containing the data 25 and 35 could be illustrated as the following:

Element Name- **myArray**	Value
myArray[0]	25
myArray[1]	35

★ Notice that the index values start at 0 and not 1.

- in addition to storing data, ***Array*** objects provide a number of useful properties and methods to manipulate the data in the array and
- find out information such as the size of the array

How to Create an Array?

UNIVERSITY of INFORMATION TECHNOLOGY

Example

```
var myArray= new Array(); // Declares a new array with no data
var myArray = new Array("One", "Two", 3);// Array with 3 elements
myArray[3] = "Four"; //  can add new  element to an existing array
myArray[5] = "Six"; // can skip the index,  but the skipped element is also created
```

Note that, as with everything in JavaScript, the code is case-sensitive, so if you type array() rather than Array(), the code won't work.

- Let's look some of the more useful property and methods of Array objects.

Finding Out How Many Elements Are in an Array — The length Property

UNIVERSITY of INFORMATION TECHNOLOGY

- The length property gives you the number of elements within an array,
- Can be used to find the index of the last element in the array

Example:

```
var names = new Array();
names[0] = "Paul";
names[1] = "Jeremy";
names[11] = "Nick";
document.write("The last name is" + names[names.length - 1]);
```

- This property is useful where a JavaScript method returns an array it has built itself.
- **For example, *split()*** method, which splits text into pieces and passes back the result as an Array, without the ***length*** property, there is no way to know what the index is of the last element in the array.

Joining Arrays — The concat() Method

- take two separate arrays and join them together into one big array.
- which is the combination of the two arrays: the elements of the first array, then the elements of the second array.
- use the method on your first array and pass the name of the second array as its parameter.

```
var names = new Array("Paul", "Jeremy", "Nick");
var ages = new Array(31,30,31);
var concatArray = names.concat(ages);
```

Element Index	0	1	2	3	4	5
Value	Paul	Jeremy	Nick	31	30	31

Copying Part of an Array —The slice() Method

- to copy a portion of an array, use the slice() method.
- using the slice() method, can slice out a portion of the array and assign it to a new variable name.
- slice() method has two parameters:
  - index of the first element you want copied
  - index of element marking the end portion you are slicing out (optional)

```
var names = new Array("Paul", "Sarah", "Jeremy", "Adam", "Bob");
var slicedArray = names.slice(1,3);
```

☞ After slicing, result table is like this:

Index	0	1
Value	Sarah	Jeremy

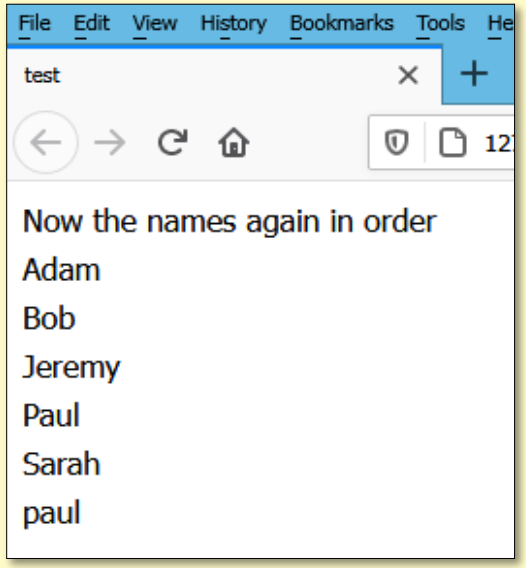
Note: The original array will not be changed.

Putting Your Array in Order — The sort() Method

- Use in alphabetical or numerical order
- In the following code, you define your array and then put it in ascending alphabetical order using names.sort().

Example:

```
var names = new Array("Paul", "Sarah", "Jeremy", "Adam" ,"paul", "Bob");
var elementIndex;
names.sort();
document.write("Now the names again in order" + "<br />");
for (elementIndex = 0; elementIndex < names.length; elementIndex++)
  {
        document.write(names[elementIndex] + "<br />");
  }
```



- Don't forget that the sorting is case sensitive, so **Paul** will come before **paul**.
- Remember that JavaScript stores letters encoded in their equivalent Unicode number, and that sorting is done based on Unicode numbers rather than actual letters.

Putting Your Array into Reverse Order — The reverse() Method

- The final method for the Array object is the reverse() method, which reverses the order of the array so that the elements at the back are moved to the front.

```
var myShopping=new Array("Eggs", "Milk", "Potatoes", "Cereal", "Banana");
myShopping.reverse();
document.write(myShopping);
```

Result:

Banana	Cereal	Potatoes	Milk	Eggs

Removing and Adding Array Elements— The pop() and push() Method

UNIVERSITY of INFORMATION TECHNOLOGY

pop()

Removes the last element from an array and returns that element.

```
var cats = new Array('Bob', 'Willy', 'Mini');
cats.pop();
document.write(cats); // ('Bob', 'Willy')
```

push()

Add one or more elements to the array and returns the array

```
cats = new Array('Bob');
cats.push('Willy','Mini');
document.write(cats); // ('Bob', 'Willy', 'Mini')
```

4.2 New Array Methods

- In 2005, Mozilla updated the JavaScript engine in Firefox.
- They added **seven new methods** to the Array object.
- These seven methods can be divided into two categories: location methods and iterative methods.
- These methods do not work Internet Explorer, however, work in Firefox, Safari, Opera, and Chrome.

Seven New Methods

- indexOf()
- lastIndexOf()

Location Methods

Similar to String Object's Methods

- every()
- some()
- filter()

Testing Methods

- forEach()
- map()

Operation Methods

Iterative Methods

Iterating Through an Array Without Loops

- remaining five methods are called iterative methods because they iterate, or loop, through the array.
- these methods execute a function you define on **every element**
- the function these methods use must accept **three arguments**

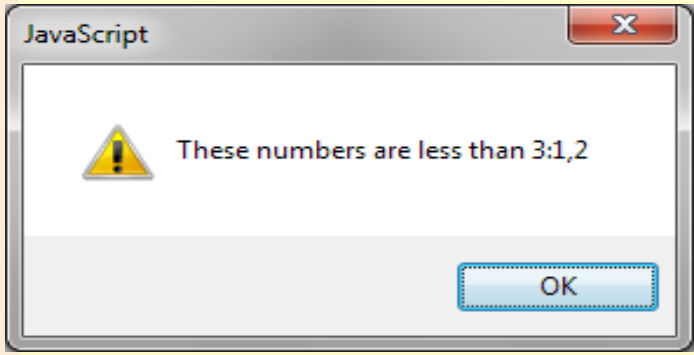
```
function functionName(value, index,
array)
{
        // do something here
}
```

  - the first is the value of the element,
  - next is the index of the element, and
  - finally is the array itself.

Testing Each Element — The every(), some(), and filter() Methods

- The every() method tests whether all elements in the array pass the test in function
- the some() method test only cares if some of the elements pass the test in function

```
var numbers = new Array(1, 2, 3, 4, 5);
function isLessThan3(value, index, array)
{
	var returnValue = false;
	if (value < 3)
	{
		returnValue = true;
	}
	return returnValue;
}
alert(numbers.every(isLessThan3));//  false
alert(numbers.some(isLessThan3));//  true
if (numbers.some(isLessThan3))
{
	var result = numbers.filter(isLessThan3);
	alert("These numbers are less than 3: " + result);
}
```



- ✓ the filter() method executes your function on every element in the array, and
- ✓ if the function returns true for a particular element, that element is added to another array the filter() method returns.

Operating on Elements — The forEach() and map() Methods

- Unlike the previous iterative methods, these two will perform some kind of operation that uses the element in some way.
  - In the following code, the value of each element is doubled, and the result is shown in an alert box using two ways: simple for loop method and forEach method.

```
var numbers = new Array(1, 2, 3, 4, 5);
for (var i = 0; i < numbers.length; i++ )
{
    var result = numbers[i] * 2;
    alert(result);
}
```

```
var numbers = new Array(1, 2, 3, 4, 5);
function doubleAndAlert(value, index, array)
{
    var result = value * 2;
    alert(result);
}
numbers.forEach(doubleAndAlert);
```

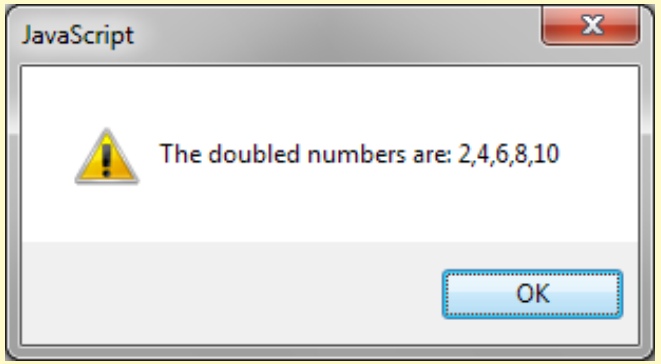
forEach() Method Cont'd

- the doubleAndAlert() function doesn't return a value like the testing methods.
- it cannot return any value; its only purpose is to **perform an operation** on every element in the array.
- while this is useful in some cases, it's almost useless when you want the results of the operation.
- that's where the map() method comes in.

map() Method

❖ the map() method is similar to that of forEach(), except that the results of every operation are stored in another array that the map() method returns.

```
var numbers = new Array(1, 2, 3, 4, 5);
function doubleAndAlert(value, index, array)
{
var result = value * 2;
return result;
}
var doubledNumbers = numbers.map(doubleAndAlert);
alert("The doubled numbers are: " + doubledNumbers);
```



➢ Instead of outputting in and alert box, can show the full result after calling **map()** method

4.3 The Math Object

- Provides a number of useful mathematical functions and number manipulation methods.
- The ***Math*** object is a little unusual in that JavaScript automatically creates it for you.
- There's no need to declare a variable as a ***Math*** object or define a ***new Math*** object before being able to use it.

For Example:

```
var myvariable=new Math();
```

Properties of Math Object

- The properties of the ***Math*** object include some useful math constants, such as the PI property
- PI property give the value 3.14159
- To access these properties, as usual, by placing a dot after the object name (**Math**) and then writing the property name
- For example, to calculate the area of a circle, use the following code:

```
var radius = prompt("Give the radius of the circle", "  ");
var area = Math.PI * radius * radius;
document.write("The area is "  + area);
```

The abs() Method

- ✓ The **abs()** method returns the absolute value of the number passed as its parameter.
- ✓ It returns the positive value of the number. So -1 is returned as 1, -4 as 4, and so on.
- ✓ However, 1 would be returned as 1 because it's already positive.

For example, the following code writes the number **101** to the page.

```
var myNumber = -101;
document.write(Math.abs(myNumber));//101
```

Finding the Largest and Smallest Numbers: the Min() and Max() Methods

UNIVERSITY of INFORMATION TECHNOLOGY

- If there are two numbers and want to find either the largest or smallest of the two, the Math object provides the **min()** and **max()** methods
- Both accept at least **two arguments**, all of which must obviously be numbers.

For example:

```
a = Math.max(b, c); //  Returns the greater of two numbers
a = Math.min(b, c); //  Returns the smaller of two numbers
var max = Math.max(21,22); //  result is 22
var min = Math.min(30.1, 30.2); //  result is 30.1
```

★ **max()** and **min()** methods can accept many numbers; **not limited to two**

Rounding Numbers

- The Math object provides a few methods to round numbers, each with its own specific purpose.

ceil() Method

floor() Method

round() Method

- The **ceil(), floor(), and round()** methods all remove the numbers after a decimal point and return just a whole number.

ceil() Method

- always rounds a number up to the next largest whole number or integer.

```
var myNumber = 101.01;
document.write(Math.ceil(myNumber) );//102
document.write(parseInt(myNumber));//101
```

- different from using the ***parseInt()*** function
- ***parseInt()*** simply chops off any numbers after the decimal point to leave a whole number

floor() Method

- removes any numbers after the decimal point, and returns a whole number or integer.
- ***floor()*** always rounds the number down.

```
var myNumber = 10.01;
document.write(Math.floor(myNumber) + “<br />”);//10
var my Number=-9.9;
document.write(Math.floor(myNumber) ;//-10
```

round() Method

- The ***round()*** method is very similar to ***ceil()*** and ***floor(),*** except that instead of always rounding up or always rounding down
- it rounds up only if the decimal part is .5 or greater, and rounds down otherwise.

```
var myNumber = 44.5;
document.write(Math.round(myNumber) + "<br />");//45
myNumber = 44.49;
document.write(Math.round(myNumber));//44
```

random() Method

- returns a random floating-point number in the range between 0 and 1, where 0 is included and 1 is not.

- very useful for displaying random banner images or for writing a JavaScript game

The following example show the rolling of a single dice for 10 times and then, 10 random numbers are written to the page.

```
<script type="text/javascript">
var throwCount;
var diceThrow;
for (throwCount = 0; throwCount < 10; throwCount++)
{
    diceThrow = (Math.floor(Math.random() * 6) + 1);
    document.write(diceThrow + "<br>");
}
</script>
```

to get 10 random number

to ensure random number be between 1 to 6

pow() Method

- The ***pow()*** method raises a number to a specified power.
- It takes two parameters, the first being the number you want raised to a power, and the second being the power itself.

Math.pow(2,8) — the result being 256

Using pow()

Example

```
<script type="text/javascript”>
function fix(fixNumber, decimalPlaces)
{
   var div = Math.pow(10,decimalPlaces);
   fixNumber = Math.round(fixNumber * div) / div;
   return fixNumber;
}
</script>
</head>
```

```
<body>
<script type="text/javascript">

 var number1 = prompt("Enter the number with decimal places you want to fix","");
 var number2 = prompt("How many decimal places do you want?","");

 document.write(number1 + " fixed to " + number2 + " decimal places is: ");
 document.write(fix(number1,number2));

</script>
```

Number Object

- ✓ As with the ***String*** object, ***Number*** objects need to be created before they can be used.
- To create a ***Number*** object, you can write the following:

```
var firstNumber = new Number(123);

var secondNumber = new Number('123');
```

- ✓ You'll look at ***toFixed()*** method of the ***Number*** object, that's the most useful method for everyday use.

toFixed() Method

- The ***toFixed()*** method cuts a number off after a certain point.
- The method's only parameter is the number of decimal places you want your number fixed to.
- The toFixed() method doesn't chop off the digits not required, it also rounds up or down.
- Can only fix a number from 0 to 20 decimal places.

Example:

```
var x = new Number(9.654);
x=x.toFixed(0);          // returns 10
x=x.toFixed(1);          // returns 9.7
x=x.toFixed(2);          // returns 9.65
x=x.toFixed(4);          // returns 9.6540
```

★ toFixed(2) is perfect for working with money.

QUIZ TIME

1

What does Math.random() do?

A Returns a random number more than 0 and less than 1.

B Randomly selects a number 1-10.

C Returns a random number more than 0 up to and including 1.

D Returns a random number from and including 0 to less than 1.