
UNOBTRUSIVE
JAVASCRIPT
CAN MAKE
HTML
DO MAGICAL THINGS
WITHOUT GOING & BLOOMIN' WELL RUINING
THE EXPERIENCE
FOR THOSE LESS FORTUNATE
creative commons

Web Technology

More about JavaScript

Last Week

- Embed Javascript on a HTML document `<script>`
- Statements e.g. `document.write("hello");`
- Blocks: `{ … }`
- Variables: `var z = x + 1;`
- Operators: `+ - * / % ++ -- = += -= *= /= &=`
- Operators Precedence: `(4 + 3) * 2`

JavaScript Loops

- Often when you write code, you want the same block of code to run over and over again in a row.
- In JavaScript, there are two different kind of loops:
  - `for` - loops through a block of code a specified number of times
  - `while` - loops through a block of code while a specified condition is true

JavaScript Functions

- To keep the browser from executing a script when the page loads, you can put your script into a function.
- A function contains code that will be executed by an event or by a call to the function.
- You may call a function from anywhere within a page (or even from other pages if the function is embedded in an external .js file).

Where to put functions?

- Functions can be defined both in the `<head>` and in the `<body>` section of a document.
- Can also use an external .js file
- Which is best?

Scripts in <head>

- Useful for small, essential scripts
- <head> is loaded by the browser before any other page elements
- Large, complex functions can cause slow page loads
  - Will every user need this script?

Scripts in <body>

- Another common place to write functions is immediately before the closing </body> tag
- Scripts are evaluated by browser at the end of the page load, meaning the user sees the page content before the script is parsed
- Useful for non-essential/time-critical scripts specific to this page
  - Like form validation
- Not so good for common page elements
  - Menu bars

.js files

- Loaded in <head>, so have a load-time impact, similar to scripts directly written there
- However, .js files can be **cached,** so they are only loaded the first time that a visitor uses your site
- **Good** for common page elements (JS for menu bars, UI widgets)
  - jQuery, Bootstrap can be cached across websites
- Can also be **minified** to save bandwidth

Minification

```
function validate()     {
       if( document.myForm.Name.value == "" )
       {
         alert( "Please provide your name!" );
         document.myForm.Name.focus() ;
         return false;
       }

       if( document.myForm.EMail.value == "" )
       {
         alert( "Please provide your Email!" );
         document.myForm.EMail.focus() ;
         return false;
       }

       if( document.myForm.Country.value == "-1" )
       {
         alert( "Please provide your country!" );
         return false;
       }
       return true;
 }
```

```
function validate(){return""==document.myForm.Name.value?(alert("Please provide your name!"),document.myForm.Name.focus(),!1):""==document.myForm.EMail.value?(alert("Please provide your Email!"),document.myForm.EMail.focus(),!1):"-1"==document.myForm.Country.value?(alert("Please provide your country!"),!1):!0}
```

- The two functions do the same thing
- Minified version is harder to read, but faster to load (50% smaller)

Minification is automated

- E.g http://jscompress.com/
- Keep your human-readable version for editing
- minified versions for production server

How to Define a Function

- A function can be defined using the following syntax:

```
function functionname(var1,var2,...,varN)
{
    some code
}
```

- The parameters `var1`, `var2`, etc. are variables or values passed into the function. The `{` and the `}` defines the start and end of the function.
- A function with no parameters must include the parentheses `()` after the function name.

Function – Example

```
<html>
<head>
<script type="text/javascript">
  function displaymessage(msg)
  {
    alert(msg);
  }
 </script>
</head>
<body>
<form>
<input type="button" value="Click me!"
              onclick="displaymessage('Hello World!')" />
</form>
</body>
```

Example available at http://jsfiddle.net/edo77uk/Y72pv/

The `return` statement

- The `return` statement is used to specify the value that is returned from the function.
- So, functions that are going to return a value must use the return statement.
- The example below returns the product of two numbers (a and b):

```
function product(a,b)
{
   return a*b;
}
document.write(product(4,3));
```

Popup Boxes

- **Alert Box** `alert("sometext");`
  - When an alert box pops up, the user will have to click "OK" to proceed.
- **Confirm Box** `var x = confirm("sometext");`
  - When a confirm box pops up, the user will have to click either "OK" or "Cancel" to proceed.
  - If the user clicks "OK", the box returns `true`. If the user clicks "Cancel", the box returns `false`.
- **Prompt Box**

```
var x = prompt("sometext","defaultvalue");
```

  - When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed after entering an input value.
  - If the user clicks "OK" the box returns the input value. If the user clicks "Cancel" the box returns `null`.

Object Oriented Programming

- JavaScript is an Object Oriented Programming (OOP) language. An OOP language allows you to define your own objects and make your own variable types.
- JavaScript has many built-in objects e.g. `String`, `Date`, `Array` and much more.
  - See: http://www.w3schools.com/jsref
- An object is just a special kind of data. An object has properties and methods.

Properties

- Properties are the values associated with an object.
- In the following example we are using the length property of the String object to return the number of characters in a string:

```
var txt="Hello World!";
document.write(txt.length);
```

- The output of the code above will be: 12

Methods

- Methods are the actions that can be performed on objects.
- In the following example we are using the `toUpperCase()` method of the String object to display a text in uppercase letters:

```
var str="Hello world!";
document.write(str.toUpperCase());
```

- The output of the code above will be:
  `"HELLO WORLD!"`

`Date` Object

- `Date` objects are used to work with dates and times.
- `Date` objects are created with the `Date()` constructor.
- There are four ways of instantiating a date:

```
new Date() // current date and time
new Date(milliseconds) //milliseconds
                       // since 1970/01/01
new Date(dateString)
new Date(year, month, day, hours,
         minutes, seconds, milliseconds)
```

- Most parameters above are optional. Not specifying, causes 0 to be passed in.
- All dates are calculated in milliseconds from 01 January, 1970 00:00:00 Universal Time (UTC) with a day containing 86,400,000 milliseconds.

Set Date

- We can easily manipulate the date by using the methods available for the Date object.
- In the example below we set a Date object to a specific date (14th January 2010):

```
var myDate=new Date();
myDate.setFullYear(2010,0,14);
```

- And in the following example we set a Date object to be 5 days into the future:

```
var myDate=new Date();
myDate.setDate(myDate.getDate()+5);
```

- **Note:** If adding five days to a date shifts the month or year, the changes are handled automatically by the Date object itself!

Date Object: get methods

- `getTime()` - Number of milliseconds since 1/1/1970 @ 12:00 AM
- `getSeconds()` - Number of seconds (0-59)
- `getMinutes()` - Number of minutes (0-59)
- `getHours()` - Number of hours (0-23)
- `getDay()` - Day of the week(0-6). 0 = Sunday, ... , 6 = Saturday
- `getDate()` - Day of the month (1-31)
- `getMonth()` - Number of month (0-11)
- `getFullYear()` - The four digit year (1970-9999)

```
var currentTime = new Date()
var month = currentTime.getMonth() + 1
var day = currentTime.getDate()
var year = currentTime.getFullYear()
document.write(month + "/" + day + "/" + year)
```

How do you write a conditional statement for executing *some code* if "i" is NOT equal to 5?

A.
```
if (i <> 5)
    { some code }
```

B.
```
if (<>5)
    { some code }
```

C.
```
if (i != 5)
    { some code }
```

D.
```
If (i =! 5) then
    { some code }
```

Array Object

- An array object can hold more than one value, at a time.
- If you have a list of items, storing the values in single variables could look like this:

```
cars1="Saab";
cars2="Volvo";
cars3="BMW";
```

- However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300? The best solution here is to use an array!
- An array can hold all your variable values under a single name. And you can access the values by referring to the array name.
- Each element in the array has its own ID so that it can be easily accessed.

Create an Array

- An array can be defined in three ways.
- The following code creates an Array object and assigns it to the variable called myCars:

```
var myCars=new Array();
myCars[0]="Saab";
myCars[1]="Volvo";
myCars[2]="BMW";
```

- Same as
  - `var myCars = new Array("Saab", "Volvo", "BMW");`
  - `var myCars = ["Saab", "Volvo", "BMW"];`

Access an Array

- You can refer to a particular element in an array by referring to the name of the array and the index number.
- The index number starts at 0.
- The following code:

```
document.write(myCars[0]);
```

- Will result in the following output: "Saab"
- Can determine number of items in an array by using length property; e.g.

```
myCars.length
```

- Will return 3

Array Indices

- It may seem strange that array indices start at 0 and not 1.

mycars	Saab	Volvo	BMW
	[0]	[1]	[2]

- The length is 3, but the last index is 2
  - myCars[myCars.length] will not work!
  - myCars[myCars.length-1] is the index of the last element
- This is for historical reasons, so we live with it.

Modify an Array

- To modify a value in an existing array, just add a new value to the array with a specified index number:

```
myCars[0]="Opel";
```

- Now, the following code line:

```
document.write(myCars[0]);
```

- will result in the following output: "Opel"

Iterate over an array

- Suppose we want all car names in uppercase
- Use the for loop:

```
var myCars = ["Saab", "Volvo", "BMW"];

for (i = 0; i < myCars.length; i++) {
   document.write(myCars[i].toUpperCase() + " ");
}
```

Iteration (2)

- Could also use the while loop:

```
var myCars = ["Saab", "Volvo", "BMW"];

var i = 0;
while (i < myCars.length) {
    document.write(myCars[i].toUpperCase() + " ");
    i++;
}
```

Prefix vs postfix

- Both `i++` and `++i` are shorthand for `i = i+1`
- `i++` gives the value of `i`, and **then** adds 1 to it (postfix)
- `++i` adds 1 to the value of `i`, **and then** returns the value (prefix)

Postfix

- Both `i++` and `++i` are shorthand for `i = i+1`
- The difference is in how the values are used:

```
var myCars = ["Saab", "Volvo", "BMW"];
i = 0;
alert (myCars[i++]) //shows Saab
```

- i is now 1

Prefix

- Both `i++` and `++i` are shorthand for `i = i+1`
- The difference is in how the values are used:

```
var myCars = ["Saab", "Volvo", "BMW"];
i = 0;
alert (myCars[++i]) // shows Volvo
```

- i is now 1

Postfix

- So, we can shorten our while loop using the postfix operator:

```
var myCars = ["Saab", "Volvo", "BMW"];

var i = 0;
while (i < myCars.length) {
    document.write(myCars[i++].toUpperCase() + " ");
}
```