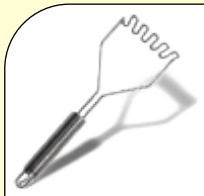
Mashups: Remixing the Web

Lecture 5: JavaScript Fundamentals



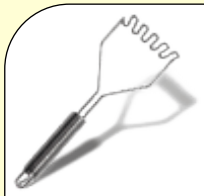
Preparing for Today's Exercises

You probably already have these installed from previous labs. You will need:

- Firefox
  http://getfirefox.com

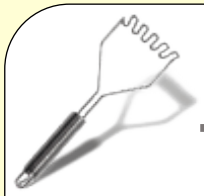
- Firebug
  http://getfirebug.com
- A Good Text Editor
  (  )

- Greasemonkey
  http://www.greasespot.net/



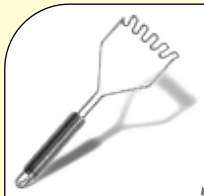
Reminders

- Assignment #2 due today!
- Assignment #3 released today, due in two weeks
- Interim course feedback form due next week: http://www.webremix.org/feedback.php
- Project team registration due in three weeks



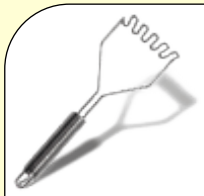
Today's Goals

- General overview of JavaScript fundamentals, including:
  - Data types
  - Syntax
  - Embedding JavaScript in a webpage
  - Functions
  - Conditionals
  - Loops
  - Event Handling
  - Traversing and modifying the DOM
- Learn to write a simple Greasemonkey script



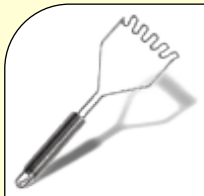
JavaScript

- JavaScript runs in the browser. It can manipulate HTML output on the screen, pull in external data, and move things around.
- JavaScript is not compiled – it's evaluated at runtime.



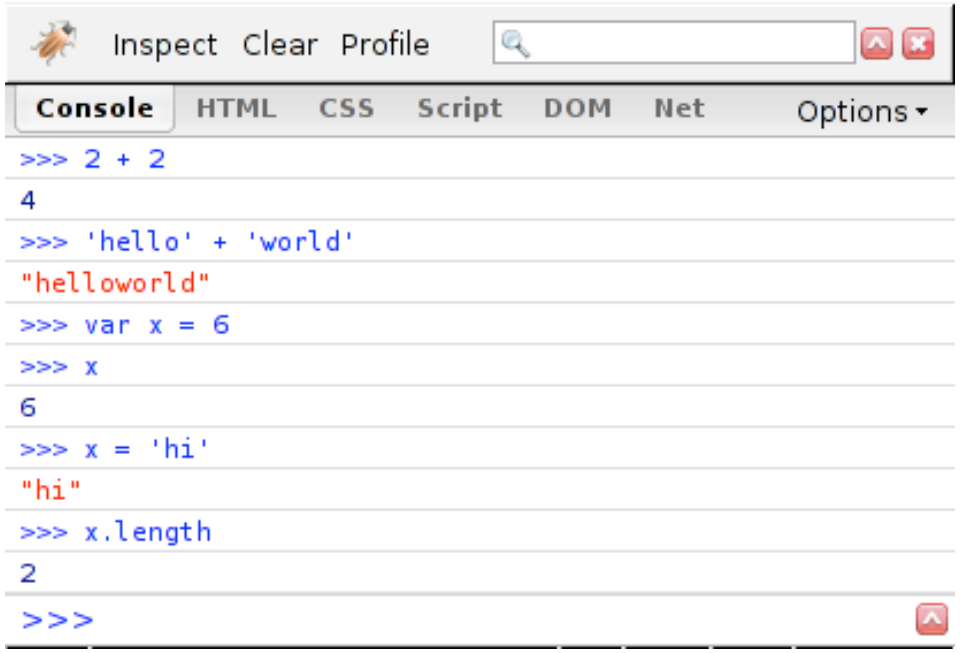
History

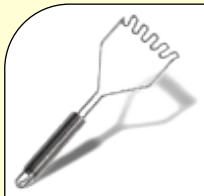
- Invented by Brendan Eich at Netscape in 1995
- Originally called LiveScript, but renamed to JavaScript for marketing reasons (its only similarity to Java is that they are both high-level programming languages)
- JavaScript is implemented slightly differently in each browser
- ECMAScript is a standardized version of JavaScript
- For many years, it was considered a “toy” language, and was used mostly for trivial effects on websites
- After the rise of AJAX Apps, JavaScript is now at the core of many websites, and is embraced by “real programmers”



Using the Firebug Console

- JavaScript is usually evaluated after being included in a website, but the Firebug extension for Firefox provides a console which lets you run arbitrary JavaScript and see the results.
- The console can be used for quick experimentation, like for the upcoming concepts.





Basic Variables

- Use `var` to declare a variable. Its initial value will be undefined.

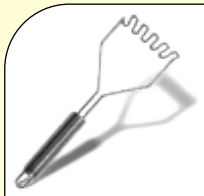
```
var x;
```

- You can assign a variable without using var, but then it will become a global variable, which you usually don't want.
- You can later initialize the variable you declared:

```
x = 5;
```

- Or you can declare and initialize all at once:

```
var x = 5;
```

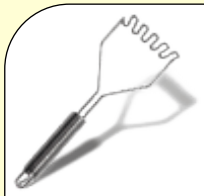


Basic Data Types

- Variables are loosely typed. They can be assigned to any data type, and can later be re-assigned to different data types:

```
var x;
x = "6";
x = 2 + 2;
x = false;
x = null;
```

- The primitive data types in JavaScript are `string`, `number`, `boolean`, `undefined`, and `null` (shown above). Anything that's not one of those types is an "object."



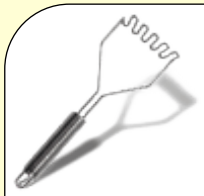
Numbers

- All numbers are 64 bit floating point – there's no differentiation between int/float/double types.
- The standard operators for numbers work in JavaScript:

```
var x = 2 * 3;
var y = x / 4;
var z = x - y;
```

- When an operation doesn't result in a valid number, `NaN` is returned.
- There are various utility functions in the Math library:

```
var x = Math.abs(-454);
var y = Math.pow(2, 3);
```



Strings

- Strings are immutable, but a new string can be created easily with concatenation:

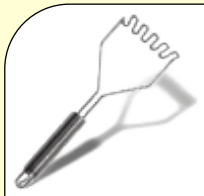
```
var x = "hello";
var y = "world";
var z = x + " " + y;
x += "!";
```

- Single quotes and double quotes are the same:

```
var x = "hi";
var x = 'hi';
```

- Single quotes can be used inside double quotes, and vice versa:

```
var x = "then he said, 'thats awesome!'";
var y = 'then he said, "thats awesome!"';
```



Strings to and from Numbers

- Strings can be converted to numbers with convenience functions or with a formal cast:

```
var x = Number("43.232");
```

- Use `parseFloat` to turn a string into a number:

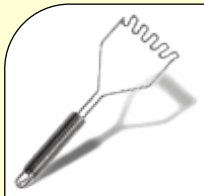
```
var x = parseFloat("43.232");
```

- Use `parseInt` to return a number with no digits after the decimal:

```
var x = parseInt("43.232");
```

- Numbers can be converted to strings with a formal cast, or just by appending an empty string.

```
var x = 23 + "";
var x = String(23);
```



String Methods

- Strings have many native helper methods:

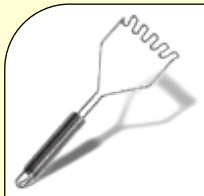
```
var greeting = "Hello there";
greeting.charAt(0); // "H"
greeting.toUpperCase(); // "HELLO THERE"
```

- This `indexOf` function returns the start index of the first occurrence of a given character or substring, or -1 if not found.

```
greeting.indexOf("e"); // 1
greeting.indexOf("there") // 6
greeting.indexOf("E") // -1
```

- Method calls can be chained in JavaScript:

```
greeting.toUpperCase().indexOf("E")
```



String Methods

- The `slice` method is a handy way to create substrings. `slice(start, end)` returns a copy of the string beginning at `start` and extending up to but not including `end`:

```
'Hello'.slice(1, 3); // "el"
```

- If no `end` is specified, it copies up to the end of the string:

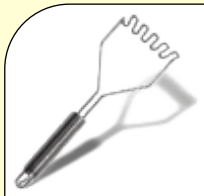
```
'Hello'.slice(2); // "llo"
```

- The `replace` method is a handy way to create a new string by replacing a substring with another substring:

```
'hi, hi'.replace('hi', 'bye'); // "bye, hi"
```

- It replaces only the first occurrence of the substring by default – specify `'g'` to have it replace every occurrence:

```
'hi, hi'.replace('hi', 'bye', 'g'); // "bye, bye"
```



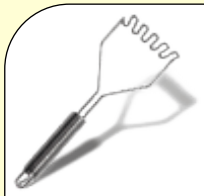
Using in a Webpage

- JavaScript can be embedded inside script tags on a page:

```
<script type="text/javascript">
var x = "Hello World";
alert(x);
</script>
```

- JavaScript can also be in external files, and referenced in a page:

```
<script type="text/javascript"
        src="external.js"></script>
```

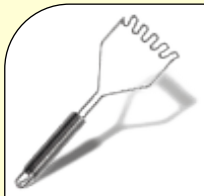


Outputting to HTML

- The absolute simplest way to modify the HTML of a page using JavaScript is with the `document.write` method.
- `document` is a predefined browser variable that always points to the current HTML document. The write method will let us insert text (or HTML) into the document immediately following the `<script>` element:

```
<script type="text/javascript">
document.write("Hello World");
</script>
```

- Note: The inserted content will not appear in the browser's "view source" window (try it out), but it will appear in the page model accessible via JavaScript (more on that later).

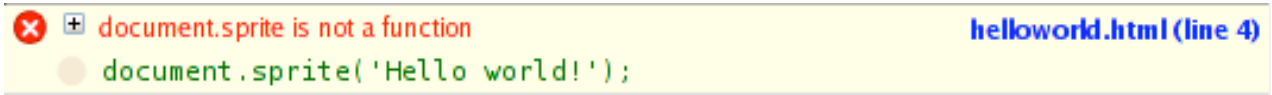


Debugging with Firebug

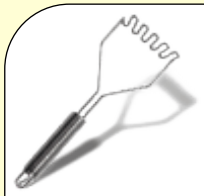
- The Firebug console will output any errors it encounters while running JS on the current page. The most common errors are trying to use functions or variables that don't exist. Try the following in an HTML file:

```
document.sprite("Hello World!");
```

- The FF status bar will display "1 Error", and you can open up the Firebug console for more information. It will often point to the line of code that caused the error, and show a trace of the functions called before the error was called.



- You can also use `console.log()` to write out information to the console from within your web page's JavaScript.



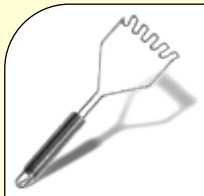
Functions

- The following code shows a function named "repeat" that takes in one argument and returns that argument repeated in a comma separated string:

```
function repeat(message) {
  var result = message + ', ' + message;
  return result;
}
```

- After (and only after) it has been defined, the function can be called like this:

```
document.write(repeat('hello'));
document.write(repeat('goodbye'));
```



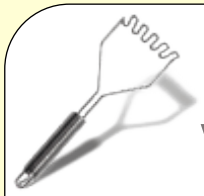
Functions

- Functions can take in any number of named parameters:

```
function multiply(number, number) {
  return number * number;
}
```

- Not all parameters need to be passed in:

```
function repeat (message, makeAllCaps) {
  if (makeAllCaps) message =
    message.toUpperCase();
  return message + "," + message;
}
repeat("hi");
repeat("hi", true);
repeat("hi", false);
```

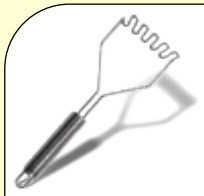


Variable Scope

- Variables defined inside a function are only known to that function, and variables defined outside a function are in the global scoped and are known everywhere.

```
var message = "Hi";
function showMessage() {
  var message = "Bye";
  alert(message);
}
showMessage();
alert(message);
```

- The first alert will show the message value local to the `showMessage` function. The second alert will show the message value in the global scope.



Functions

- Functions return either an explicit value or `undefined`:

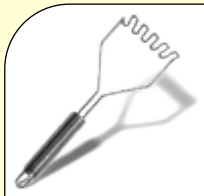
```
function doAnnoyingAlert(message) {
  alert(message.toUpperCase());
}
```

- Functions that return nothing are usually called like this:

```
doAnnoyingAlert();
```

- A function can only have one definition. The last definition is the winner:

```
function doAlert() { alert("hi"); }
function doAlert() { alert("bye"); }
doAlert();
```

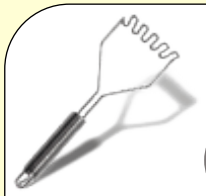


Conditionals

- The following if statement writes out various messages depending on the value of the parameters passed in:

```
function stopForSpeeding(speed, mood) {
  if (speed >= 80) {
    document.write('License and registration please.<br/>');
    if (mood == 'terrible' || speed >= 100) {
      document.write('You have the right to remain silent.<br/>');
    } else if (mood == 'bad' && speed >= 90) {
      document.write(
        'I\'m going to have to write you a ticket.<br/>');
    } else {
      document.write('Let\'s try to keep it under 80 ok?<br/>');
    }
  }
}
```

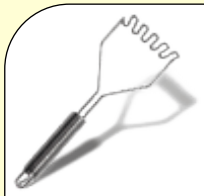
- The condition goes in parentheses after the if keyword and can include all the usual comparison operators: `==`, `!=`, `<`, `<=`, `>`, `>=`. The `else if` clause can be repeated any number of times (including `0`), and the `else` clause is optional.



Conditionals

- The typical condition evaluates to a boolean variable (`true` or `false`), but conditions can also use other data types. The following values are treated as false: undefined variables, `null`, `0`, and the empty string. Everything else counts as true.
- This is useful for checking optional arguments:

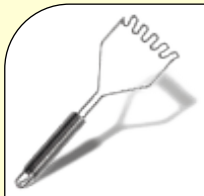
```
function repeat(message, loud) {
  var result = message + ', ' + message;
  if (loud) result += '!!!';
  return result;
}
repeat('hello', true);
repeat('bye', false);
repeat('bye');
```



Boolean Operators

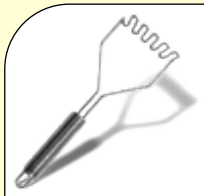
- Both && and || are "short-circuited," which means that they only evaluate their second argument if it could affect the final answer.
- This comes in useful when you want to call a method on an object that may be null or undefined:

```
if (message &&
    message.indexOf('hello') != -1) {
  ...
}
```



Exercise Time!

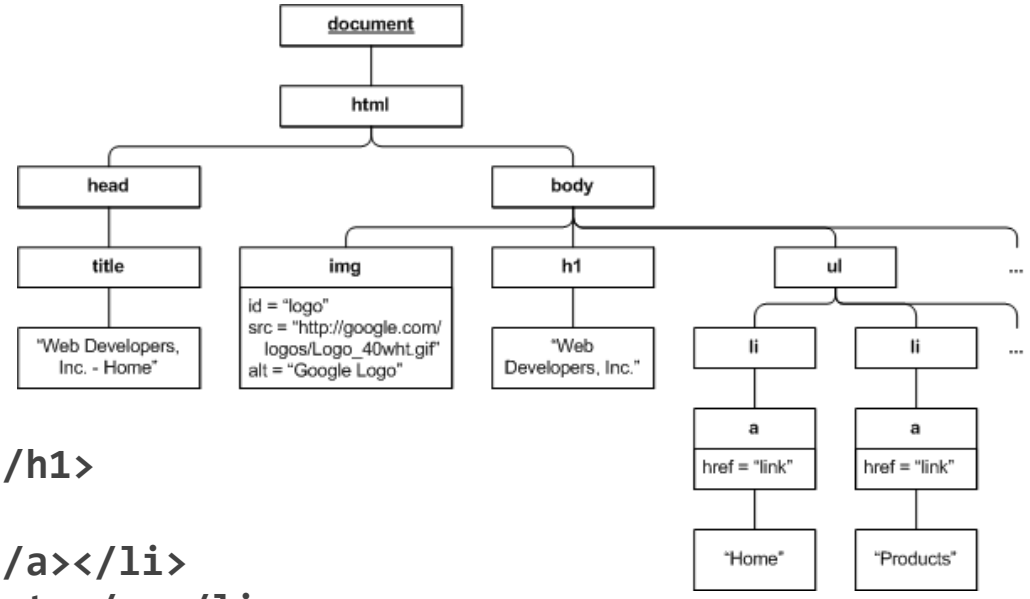
- Open up `strings.html` in the lab directory, and we'll get started.

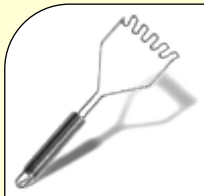


DOM

- The HTML of a webpage is exposed to JavaScript code through a structure called the Document Object Model (DOM).
- A sample webpage and corresponding DOM are shown here:

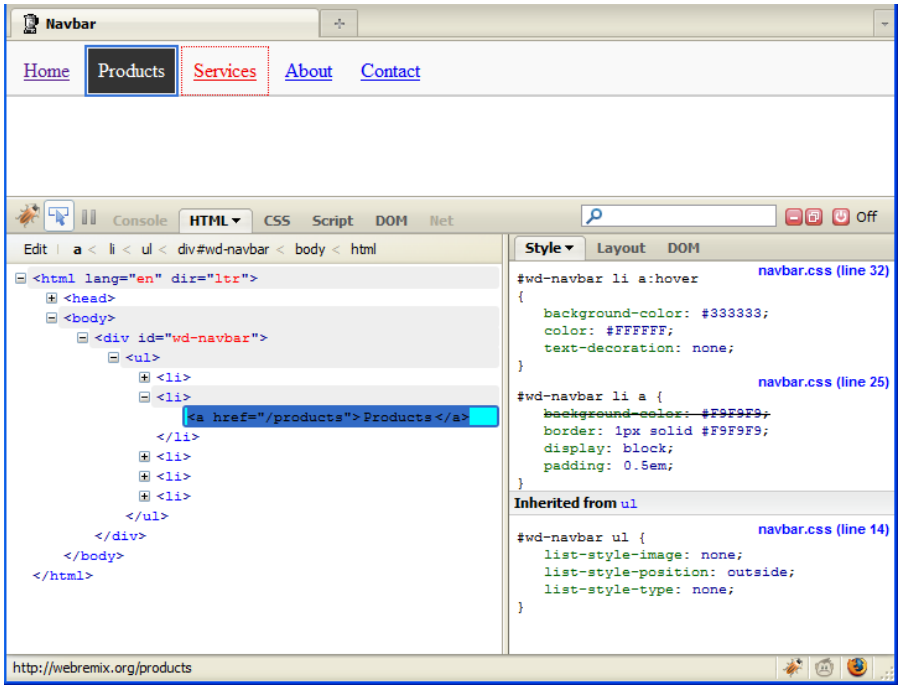
```
<html>
 <head>
  <title>Home</title>
 </head>
 <body>
  <img
      id="logo"
      src="Logo_40wht.gif"
      alt="Google Logo"/>
  <h1>Web Developers, Inc.</h1>
  <ul>
   <li><a href="link">Home</a></li>
   <li><a href="link">Products</a></li>
...
```

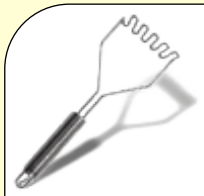




DOM in Firebug

- You can see the DOM tree in indented form in Firebug's HTML tab. Each element can be expanded to see its children.
- You can peek into the DOM objects to see the attributes available to JavaScript by selecting the DOM view in the right pane.
- The HTML tab shows the entire tree, while the DOM view shows the details of a single element.
- Hover over an element in the tree to highlight it on the page. Click "Inspect" and then click the desired element to focus it in the tree.

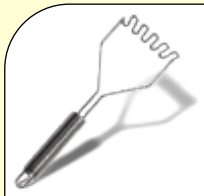




DOM Functions: getElementById

- The DOM can be programmatically traversed by calling various functions on a DOM node.
- The global `document` object is the root node of the DOM tree on every page.
- The fastest way to access one DOM node beneath the root node is to assign it an id, and use `document.getElementById("id")`, like this:

```
<img id="logo" src="Logo_40wht.gif"/>
document.getElementById('logo');
```

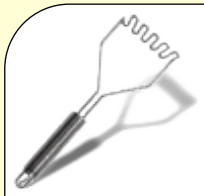


DOM Attributes

- Once you have a reference to a DOM object, you can set and get all of its standard HTML and CSS attributes.

```
var logo =
  document.getElementById('logo');
var oldSrc = logo.src;
logo.src = 'Logo_60wht.gif';
```

- When you set a new value for the `src` attribute, the browser immediately loads the new image and reflows the page.



DOM Style Attribute

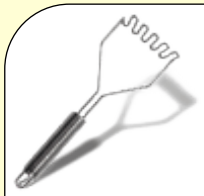
- You can set CSS properties through a special `style` attribute on every DOM node.

```
logo.style.display = 'none'
```

This immediately makes the logo disappear.

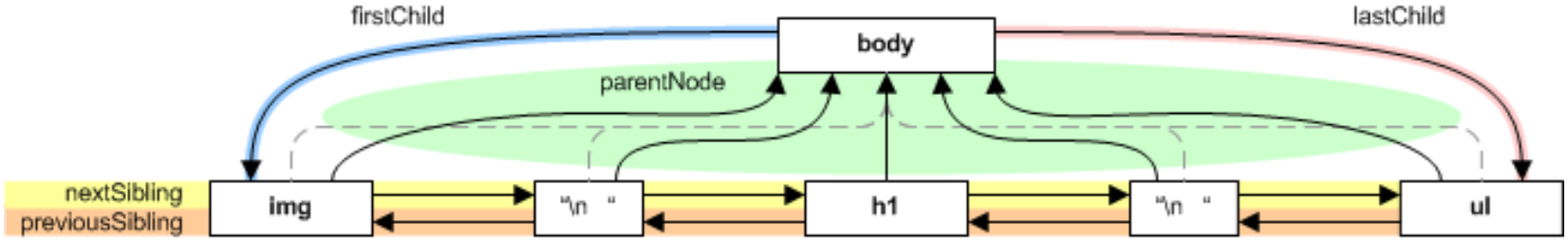
- Some property names must be adapted to become valid JavaScript identifiers. For example, `margin-top` becomes `marginTop` and class becomes `cssClass`.
- Numeric properties must be set using proper CSS syntax with their units.

```
logo.style.marginTop = '10px'; // CORRECT
logo.style.marginTop = 10;     // INCORRECT!
```

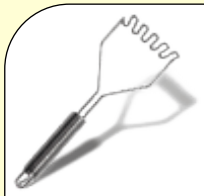


DOM Traversal

- Each DOM node has links to its neighbors (siblings, children, and parent).



- As it turns out, these neighbor links aren't very useful, as the slightest change in the page's layout (even the addition of whitespace) will change the links.
- The only link that is always reliable is `parentNode`, since text nodes (whitespace or otherwise) are always leaves in the tree.



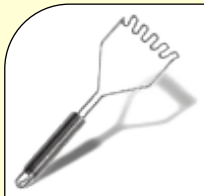
Modifying the DOM

- Every node in the DOM tree has an innerHTML attribute that contains all of its children serialized back into HTML format.

```
var nav = header.nextSibling.nextSibling;
var oldNavHTML = nav.innerHTML;
var oldHeaderHTML = header.innerHTML;
```

- You can set the `innerHTML` attribute to replace the contents of any node.

```
header.innerHTML = "Web Vandals, Ltd."
```



Modifying the DOM

- Replace text contents with different text:

```
header.innerHTML = "Web Vandals, Ltd."
```

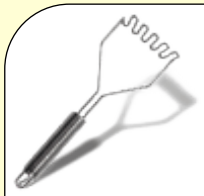
- Replace text contents with mix of text and elements:

```
header.innerHTML =
  "Web <em>Vandals</em>, Ltd."
```

- Append an element to the existing list:

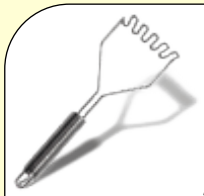
```
nav.innerHTML = nav.innerHTML +
  '<li><a href="link">Privacy</a></li>';
```

- For finer-grained insertions and deletions, it's probably best to use DOM-based methods like `insertChild`, `removeChild`, and `createElement`.



Greasemonkey

- Greasemonkey is an add-on for Firefox that automatically runs custom scripts when you load pages into your browser.
- The scripts can access and modify the page being loaded as if though they had been included in the page itself. This allows you to customize web pages even when you don't have permission to change the original at the source.
- Let's try an example!



Arrays

- A JavaScript array is a list of values (or objects). You can add and remove values from an array, and the values it holds do not all need to be of the same type.
- Create a new array with a set of items:

```
var a = ['hi', 1, 2];
```

- Get the length of the array:

```
var arrayLength = a.length;
```

- Get the item at the given index (non-negative):

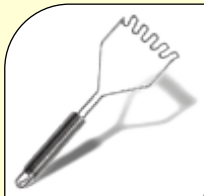
```
var firstItem = a[0];
```

- Replace the item at the given index (different type OK):

```
a[1] = 'bye';
```

- Return an new array that's a slice of the original array:

```
var slicedArray = a.slice(0, 2);
```



Arrays: Adding and Removing Values

- You can use assignment to add values to the array, but you have to be careful: the array will automatically be expanded to hold its last value, and any spots between the array's previous last value and the new one will be filled with `undefined`.
- Add an item at the given index, and grow the array:

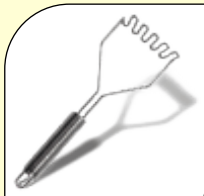
```
a[3] = false;
```

- Grow the array to length of 7:

```
a[6] = 'foo';
```

- The newly grown array will look like:

```
["hi", "bye", 2, false,
  undefined, undefined, "foo"]
```



Arrays: Adding and Removing Values

- Instead of specifying an index to add or remove values, you can use the `push` and `pop` methods:
- Create a new empty array:

```
var a = [];
```

- Add an item to the end of the array:

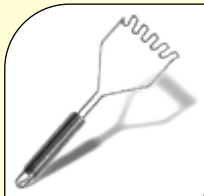
```
a.push('bar');
```

- Adds multiple items to the end of the array:

```
a.push('baz', 42);
```

- Removes and returns the last item from the array:

```
var lastItem = a.pop();
```



Arrays: Adding and Removing Values

- To add or remove values inside an array you must use `splice`.
  - The first argument specifies the target index in the array
  - The second argument is the number of items to remove, starting at that index.
  - The third argument is an array of items to insert at that index.
  - To only remove items, omit the third argument.
  - To only insert items, use 0 as the second argument. Any removed items are copied into an array and returned.
- Remove and return 2 items starting at index 1:

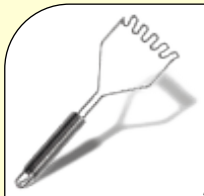
```
a.splice(1, 2);
```

- Insert items at index 1:

```
a.splice(1, 0, ['fie', 'foo']);
```

- Replace 2 items at index 1 with new items:

```
a.splice(1, 2, ['bar']);
```



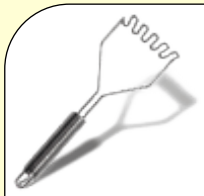
Arrays to and from Strings

- You can use `split` to transform strings into arrays, and `join` to transform arrays into strings.
- `split` extracts an array of substrings, dividing them at (and discarding) the specified separator. It can be useful for parsing structured data:

```
var words = 'write-once-read-many';
var wordsArray = words.split('-');
```

- `join` takes an array of objects and glues them together in a string with the specified separator string between each one. You can use it to format an array of data for output.

```
var stepsArray = ['start', 'middle', 'end'];
var steps = stepsArray.join(' => ');
```

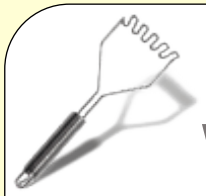


For Loops

- To iterate through an array you normally use a for loop like this:

```
var a = ['foo', 'bar', 'baz'];
for (var i = 0; i < a.length; i++) {
  // do something with a[i] here, e.g.:
  document.write(i + ": " + a[i]);
}
```

- A `for` loop statement contains 3 parts separated by semicolons:
  - `var i = 0` is the loop variable declaration and initialization.
  - `i < a.length` is the loop condition. It is checked at the top of the loop at each iteration, and the loop ends when the condition becomes false. In this case we keep iterating as long as the index is strictly smaller than the length of the array – indices are `0`-based, so the last valid index is `a.length - 1`.
  - `i++` is the loop increment. It is executed at the bottom of the loop at each iteration, and changes the value of the loop variable. In this case we simply increment `i` by `1`.

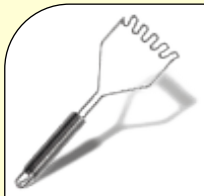


While Loops

- You could write the same loop using `while` instead:

```
var a = ['foo', 'bar', 'baz'];
var i = 0;
while(i < a.length) {
  // do something with a[i] here, e.g.:
  document.write(i + ": " + a[i]);
  i++;
}
```

- The main difference is that the loop variable is declared *outside* the loop proper, and will thus survive after the loop ends. Also, the increment is often inside the loop block, so using `continue` and `break` (which work just like in C++ and Java) can get trickier.

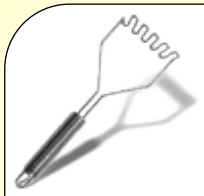


Smarter DOM Traversal

- The `document.getElementsByTagName()` method returns an array-like list of all the DOM elements with the given tag name. You can then iterate over them to do something to each one, or further refine your search.
- This code finds all links on a page and rewrites them to point to the 1999 version of the linked page:

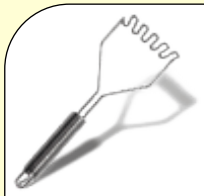
```
var links = document.getElementsByTagName('a');
for (var i = 0; i < links.length; i++) {
   if (a.href) {
    a.href = "http://web.archive.org/web/1999/" + a.href;
    a.innerHTML = "Party like it's 1999! " + a.innerHTML;
  }
}
```

- You can also call `getElementsByTagName()` on an element, in which case it will return only *descendants* with the given tag name.



Exercise Time!

- In the next exercise, we'll be working with arrays. Open up `arrays.html` to get started.

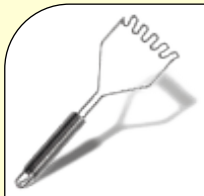


Events

- Each time something interesting happens in the browser (like the user clicks a button or the browser finishes loading the page), an event is fired.
- Simple page:

```
<html>
<head><title>Logo of Fortune</title></head>
<body>
  <img id="logo" src="logo.gif"/>
  <button id="lucky">I'm feeling lucky!
  </button>
</body>
</html>
```

- With just the HTML, clicking the button won't make anything happen. But with just a few lines of code…

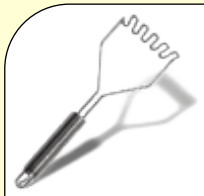


Events

- To get the button click to do something, define a function to switch the function and add code to hook it up to the button:

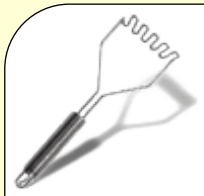
```
function switchImage() {
  document.getElementById('logo').src = "july4th08.gif";
}
document.getElementById('lucky').onclick = switchImage;
```

- By assigning the `switchImage` function to the button's `onclick` attribute, the browser will call that function each time the button is clicked. We're not calling `switchImage` ourselves; we're storing a reference to it in the `onclick` attribute for the browser to call later.
- Note: The code must come *after* the two HTML elements it references, since scripts are executed as they occur in the document, and DOM nodes that follow a script don't yet exist.



Events

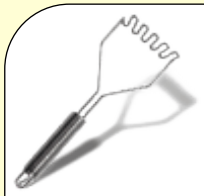
- Browsers fire many different events. Here are the most common:
  - `onclick`: Fired when an element is clicked on (or otherwise activated), including links, buttons and images.
  - `onmouseover`, `onmouseout`: Fired when the mouse enters or leaves the boundaries of an element. Useful for rollovers, etc.
  - `onmousedown`, `onmouseup`: Fired when the mouse button is pressed or released on an element. Useful if you want to do something while the mouse button is down.
  - `onload`: Available only on the `<body>` element and window object, fired when the document has finished loading. Useful in scripts that you load from an external file using a `<script src="...">` element in the document's `<head>` section, but that require the document to be loaded (and the DOM to be fully formed) before they can run.



Events

- Remember that you can access variables available in the lexical scope inside your functions, so it's easy to create an event that increments according to a global counter:

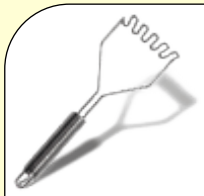
```
var logoUrls = [
  'chagall.gif', 'july4th08.gif', 'summersolstice08.gif',
  'spring08.gif', 'persian_newyear08.gif'
];
for (var i = 0; i < logoUrls.length; i++) {
  logoUrls[i] =
    'http://www.google.com/logos/' + logoUrls[i];
}

var logoIndex = 0;
function switchImage() {
  document.getElementById('logo').src =
   logoUrls[logoIndex];
  logoIndex = (logoIndex + 1) % logoUrls.length;
}
```



Events

- Every event listener is passed an event object that contains information about the event. One of its properties is the target, which contains a reference to the DOM node that fired the event.
- You can use the target to easily change attributes of that node:

```
function switchImage(event) {
  document.getElementById('logo').src =
    logoUrls[logoIndex];
  logoIndex =
    (logoIndex + 1) % logoUrls.length;
 event.target.value =
   "Showing Logo" + logoIndex;
}
```

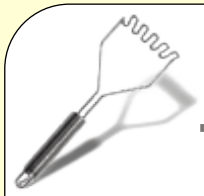


Preloading Images

- When an image is dynamically loaded in JavaScript, it can sometimes take a few seconds to show up (depending on its size). So when you're doing `mouseover` or animation effects with images, you want to preload them. You can preload by inserting hidden `img` nodes into the DOM – the browser will still load the image, and put it in its cache.

```
for (var i = 0; i < logoUrls.length; i++) {
  logoUrls[i] =
    'http://www.google.com/logos/' + logoUrls[i];
  document.write(
    '<img src="' + logoUrls[i] +
    '"style="display:none"/>');
}
```

- Note that this could also be accomplished in plaing HTML (without any JavaScript).

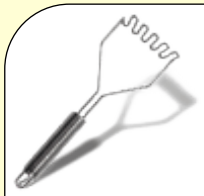


Timing and Animation

- To cause a delay between function calls (or "frames") in JavaScript you can use `window.setTimeout()`.
  `window` is a global object like `document` that refers to the JavaScript's browser environment.
  `setTimeout()` takes two parameters: a function reference to call, and the number of milliseconds to wait before calling it.

```
function switchImage() {
  document.getElementById('logo').src =
    logoUrls[logoIndex];
  logoIndex = (logoIndex + 1);
  if (logoIndex < logoUrls.length) {
    window.setTimeout(switchImage, 500);
  }
}
```

- As long as it hasn't displayed all the logos yet, this function schedule s itself to be called every 1/2 second.



More Resources

- "JavaScript: The Definitive Guide" by David Flanagan is a great desktop reference for all things JavaScript.
  Grab the latest edition (5th edition as of this writing).
- W3Schools is a reference site for JavaScript, CSS, and HTML, with lots of sample code snippets.
- Mozilla Developer Center
  has articles and documentation on all aspects of web programmi ng, both generic and Firefox-specific.
  Its coverage of JavaScript features and functions is particularly in-depth.
- QuirksMode has up-to-date cross-browser compatibility tables and in-depth articles on compatibility issues.
- BrainJar.com
  has some good introductory articles about CSS and JavaScript, al ong with fun non-trivial examples.