JavaScript On a Web Page

Agenda

- A crash course on Web programming
- The DOM
- Changing the document
- Browser Events
- Changing style from JavaScript

JavaScript On a Web Page

- JavaScript statements can be coded on a web page using three different techniques:
  - Place JavaScript code as part of an HTML element
  - Place JavaScript code between <script> tags
  - Place JavaScript code in a separate file
    - and add a reference in the src attribute of a <script> tag

JavaScript code as part of an HTML element

- Html elements have event-attributes that take JavaScript code as their value

```
<button onclick="alert('Boom!');">
DO NOT PRESS
</button>
```

Eventhandler written in JavaScript

Works but bad style - Mixes presentation with logic!

JavaScript: Using The script Element

- The script element
  - A container tag
  - May be placed in either the head or the body section of a web page
    - **It is recommended to place it as the last statement before the closing body tag**

```
<body>
  . . .
  <script>
    alert("Welcome to Our Site");
  </script>
</body>
```

JavaScript: in a separate file

- The script element
  - A container tag
  - May be placed in either the head or the body section of a web page

```
<script src="url" ></script>
```

- The loading and processing of the page pauses while the browser fetches and executes the file
- The content between the <script src="*url*"> and the </script> should be blank
- It is better to call for the script as late as possible, so that the loading of images and other components will not be delayed
- This can improve the perceived and actual page loading time. So it is usually best to make all <script src="*url*"></script> the last features before the </body>

JavaScript On a Web Page

- Works with the objects associated with a Web page by use of the environment variables like
  - window
  - document

- Executes in a sandbox
  - browsers severely limit the things a JavaScript program may do
  - it can’t modify anything not related to the web page it is embedded in

Common Uses of JavaScript

- Display a message box
- Select list navigation
- Edit and validate form information
- Create a new window with a specified size and screen position
- Image Rollovers
- Status Messages
- Display Current Date
- Calculations

The open Method

- Takes a URL as an argument, and will open a new window showing that URL

```
var perry = window.open("http://www.pbfcomics.com/257/");
```

- Because open is a method on the window object, the window. part can be left off

```
var perry = open("http://www.pbfcomics.com/257/");
```

- An opened window can be closed with its close method

```
perry.close();
```

The open Method

- The value returned by window.open is a new window
  - This is the global object for the script running in that window, and contains all the standard things like the Object constructor and the Math object
  - But if you try to look at them, most browsers will (probably) not let you!

- The exception to this rule is pages opened on the same domain
  - When a script running on a page from myDomain.net opens another page on that same domain, it can do everything it wants to with this page

Prompts

- prompt() method
  - Displays a message and accepts a value from the user

```
myName = prompt("prompt message");
```

  - The value typed by the user is stored in the variable myName

JavaScript & Accessibility

- Don’t expect JavaScript to always function for every visitor
  - Some may have JavaScript disabled
  - Some may be physically unable to click a mouse

- Provide a way for your site to be used if JavaScript is not functioning
  - Plain text links
  - E-mail contact info

THE DOM

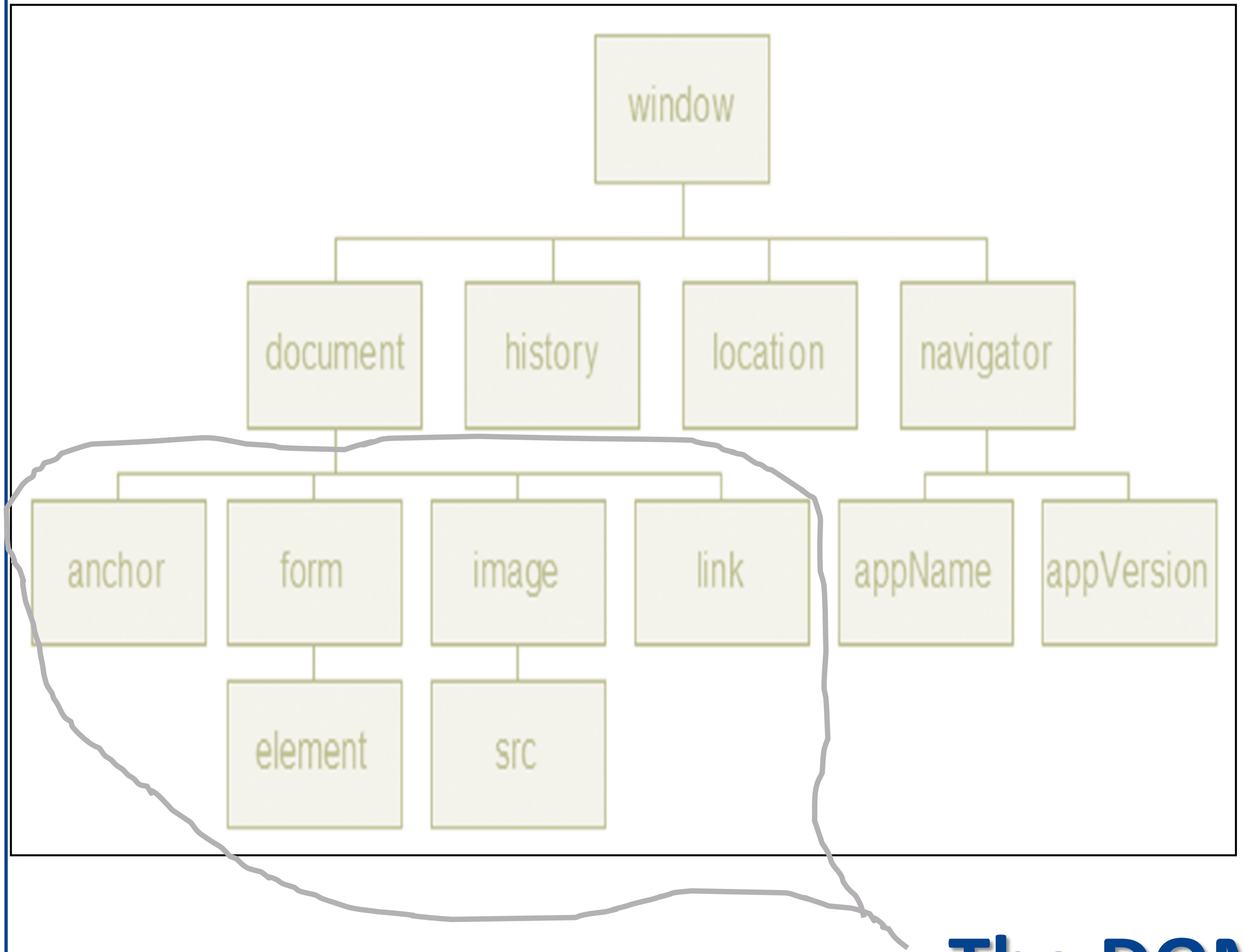
The document

- Every window object has a document property, which contains an object representing the document shown in that window
- This object contains, for example, a property location, with information about the URL of the document.

```
alert(document.location.href);
```

- Setting document.location.href to a new URL can be used to make the browser load another document

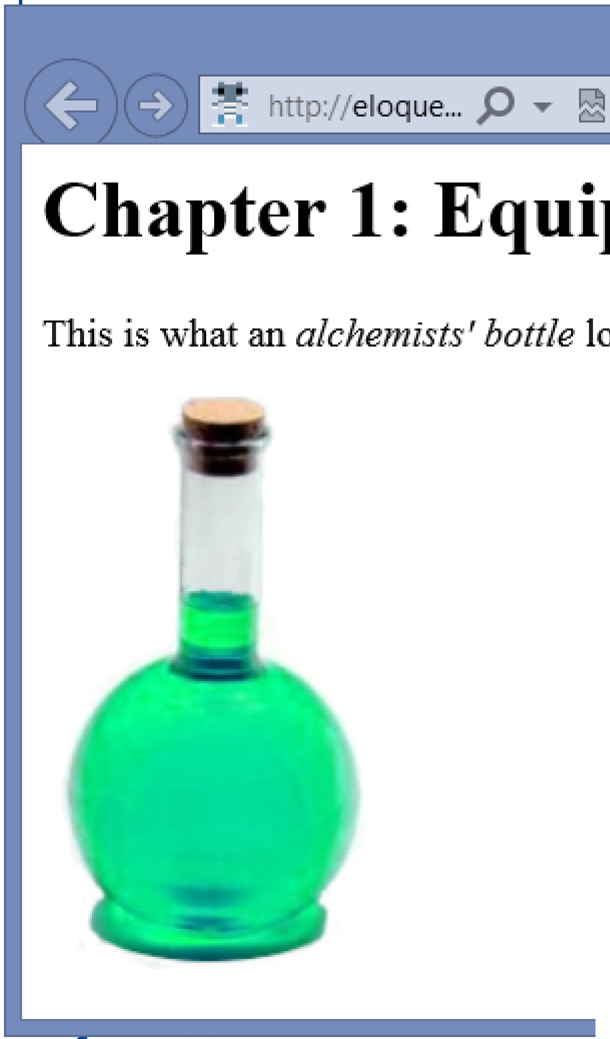
Document Object Model (DOM)



- A portion of the DOM is shown at the left
- Hierarchical structure
- Every tag of the document is represented in this model, and can be looked up and interacted with

The DOM

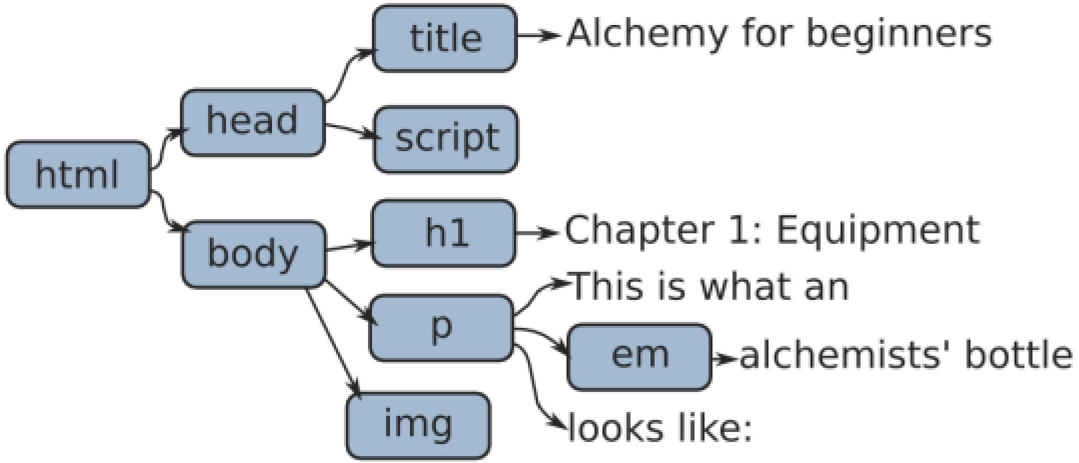
DOM Example



Chapter 1: Equipmen

This is what an *alchemists' bottle* looks like:

```html
<html>
  <head>
    <title>Alchemy for beginners</title>
    <script type="text/javascript"
src="js/chapter/dom.js"></script>
    <script type="text/javascript"
src="js/FunctionalTools.js"></script>
    <style type="text/css">
      td, th {border: 1px solid black; padding: 3px;}
      table {border-collapse: collapse;}
    </style>
  </head>
  <body><h1>Chapter 1: Equipment</h1>
    <p>This is what an <em>alchemists' bottle</em>
looks like:</p>
    <img src="img/florence_flask.png" alt="a fat
bottle" id="picture"/></body>
</html>
```



Navigating the DOM

- **`document.body`** points to the body part of the document
- The links between these nodes are available as properties of the node objects
- Every DOM object has a **`parentNode`** property, which refers to the object in which it is contained, if any
- These parents also have links pointing back to their children – in a pseudo-array called **`childNodes`**
  - there are also links called **`firstChild`** and **`lastChild`**
- Finally, there are properties called **`nextSibling`** and **`previousSibling`**, which point at the nodes sitting 'next' to a node
  - nodes that are children of the same parent

nodeType

- To find out whether a node represents a simple piece of text or an actual HTML node, we can look at its nodeType property
- This contains a number:
  - 1 for regular nodes
  - 3 for text nodes

```
function isTextNode(node) {
    return node.nodeType == 3;
}

show(isTextNode(document.body));
show(isTextNode(document.body.firstChild.firstChild));
```

Finding Elements - TagName

- We can access a node by use of childNodes and a series of `nextSibling` etc.
  - This can work, but it is verbose and easy to break
- But all element nodes have some very convenient helper functions for finding element nodes

```
var link = document.body.getElementsByTagName("a")[0];
console.log(link.href);
```

Get the first link (a-node) in the document

Finding Elements - id attribute

- Give elements that you need to have access to an id attribute and use `getElementById`

```
var picture = document.getElementById("picture");
alert(picture.src);
picture.src = "img/ostrich.png";
```

- Because document.getElementById is a ridiculously long name for a very common operation, it has become a convention among JavaScript programmers to aggressively abbreviate it to $

```
function $(id) {
    return document.getElementById(id);
}
alert($("picture"));
```

Finding Elements - class name

- `getElementsByClassName`
  searches through the contents of an element node and retrieves all elements that have the given string in their class attribute
  - Similar to getElementsByTagName

querySelector

- The `querySelector` method is useful if you want a specific, single element. It will return only the first matching element or null if no elements match
- The `querySelectorAll` method returns an array-like object containing all the elements that it matches
- Both methods takes a selector string with the same syntax as used in CSS3

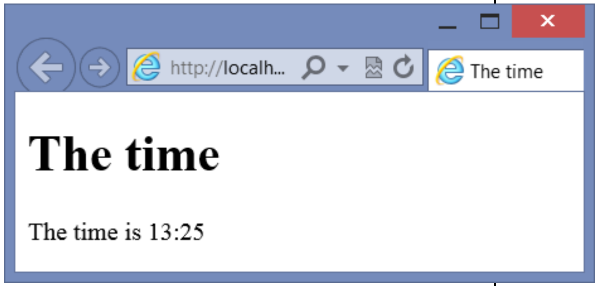
```
var nodes = document.querySelectorAll("p .animal");
```

CHANGING THE DOCUMENT

document.write

- Writes some HTML to the document
- When it is used on a fully loaded document, it will replace the whole document☹
- But if a script call it while the document is being loaded, the written HTML will be inserted into the document at the place of the script tag that triggered it
  - This is a simple way to add some dynamic elements to a page

```
</head>
<body>
    <h1>The time</h1>
    <p>The time is
      <script type="text/javascript">
          var time = new Date();
          document.write(time.getHours() + ":" + time.getMinutes());
      </script>
    </p>
</body>
</html>
```



The time

The time is 13:25

innerHTML

- The innerHTML property can be used to retrieve the HTML text *inside* of the node, without the tags for the node itself

```
alert(document.body.innerHTML);
```

- Setting the innerHTML of a node or the nodeValue of a text-node will change its content

```
document.body.firstChild.nextSibling.innerHTML =
  "Chapter 1: The deep significance of the bottle";
```

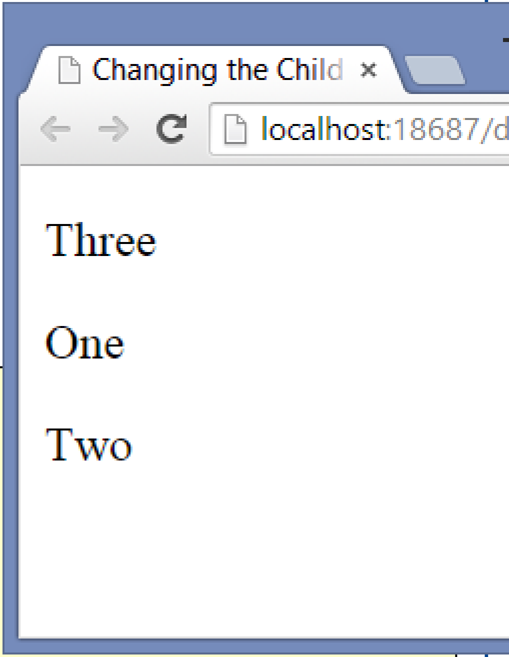
```
document.body.firstChild.nextSibling.firstChild.nodeValue =
  "Chapter 1: The deep significance of the bottle";
```

Changing the Child Nodes

- Element nodes have a number of methods that can be used to change their content
  - appendChild
  - insertBefore
  - replaceChild
  - removeChild

```
<p>One</p>
<p>Two</p>
<p>Three</p>

<script>
  var paragraphs = document.body.getElementsByTagName("p");
  document.body.insertBefore(paragraphs[2], paragraphs[0]);
</script>
```



Three

One

Two

A node can exist in the document in only one place!

Creating nodes

- `createElement()` creates a new type1 node
- `createTextNode()` creates a new type3 node

```
var secondHeader = document.createElement("h1");
var secondTitle = document.createTextNode("Chapter 2: Deep
magic");
secondHeader.appendChild(secondTitle);
document.body.appendChild(secondHeader);

var newImage = document.createElement("img");
newImage.setAttribute("src", "../img/ostrich.png");
document.body.appendChild(newImage);
```

Attributes

- Some element attributes can be accessed through a property of the same name on the element’s DOM object

```
var link = document.body.getElementsByTagName("a")[0];
link.href = "http://ase.au.dk";
```

- But HTML allows you to set any attribute you want on nodes by use of:
  - `setAttribute`
  - `getAttribute`

```
<p data-classified="secret">The launch code is 00000000.</p>

<script> var paras = document.body.getElementsByTagName("p");
if (paras[0].getAttribute("data-classified") == "secret")
  paras[0].parentNode.removeChild(para);
```

BROWSER EVENTS

You have power over your mind—not outside events. Realize this, and you will find strength.

Marcus Aurelius, *Meditations*

JavaScript and Events

- Events:
  actions taken by the web page visitor
  - clicking (click),
  - placing the mouse on an element (mouseover),
  - removing the mouse from an element (mouseout),
  - loading the page (load),
  - unloading the page (unload),
  - etc.

- The `addEventListener` function registers its second argument to be called whenever the event described by its first argument occurs

```
addEventListener("click", function() {
    console.log("You clicked!");
  });
```

Events

Event	Event Handler
click	onclick
load	onload
mouseover	onmouseover
mouseout	onmouseout
submit	onsubmit
unload	onunload

JavaScript and Events

- JavaScript can be configured to perform actions when events occur
  - The event name is coded as an attribute of an HTML tag
  - The value of the event attribute contains the JavaScript code

Example:

Display an alert box when the mouse is placed over a hyperlink

```
<a href="home.htm"
   onmouseover="alert('Click to go home')">
   Home
</a>
```

Obtrusive JavaScript - not recommended!

Unobtrusive JavaScript

- Event subscriptions are made in JavaScript

```
...
<body>
  <!-- button has no event wire up code -->
  <a id="homelink" href="home.htm">Home</a>

    <script>
       document.getElementById("homelink")
               .addEventListener("mouseover", function () {
            alert('Click to go home');
       });
    </script>
</body>
```

Best Practice

Event objects

- Event handler functions are passed an argument: the event object
  - Gives us additional information about the event
  - The information stored in an event object differs per type of event

```
<button>Click me any way you want</button>
<script>
 var button = document.querySelector("button");
 button.addEventListener("mousedown", function(event) {
   if (event.which == 1)
     console.log("Left button");
   else if (event.which == 2)
     console.log("Middle button");
   else if (event.which == 3)
     console.log("Right button");
  });
</script>
```

Event Bubbling

- An unhandled event will 'bubble' through the DOM tree
  - If you click on a link in a paragraph, any handlers associated with the link are called first
  - If there are no such handlers
    - or these handlers do not indicate that they have finished handling the event
  - Then the handlers for the paragraph, which is the parent of the link, are tried
  - After that, the handlers for document.body get a turn
  - Finally, if no JavaScript handlers have taken care of the event, the browser handles it

- The event is said to *propagate* outward, from the node where it happened
- An event handler can call the `stopPropagation` method on the event object to prevent handlers “further up” from receiving the event

Default actions

- Many events have a default action associated with them
  - E.g.: if you click a link, you will be taken to the link’s target
- Most JavaScript event handlers are called *before* the default behaviour is performed
- If the handler doesn’t want the normal behaviour to happen
  - use the `preventDefault` method on the event object
- But depending on the browser, some events can’t be intercepted
  - On Chrome keyboard shortcuts to close the current tab (Ctrl-W or Command-W) cannot be handled by JavaScript

```
var link = document.querySelector("a");
link.addEventListener("click", function(event) {
  // DoSomeThing();
  event.preventDefault();
});
```

Mouse, Touch, and Pointer Events

	Mouse Events	Touch Events	Pointer Events
Supports mouse	🟩	🟧	🟩
Supports single-touch	🟧	🟩	🟩
Supports multi-touch	🟥	🟩	🟩
Supports pen, Kinect, and other devices	🟧	🟥	🟩
Provides over/out/enter/leave events and hover	🟩	🟥	🟩
Asynchronous panning/zooming initiation for HW acceleration	🟥	🟥	🟩
W3C specification	🟩	🟩	🟩
Usable cross-browser on mobile devices	🟧	🟩	🟥
Usable cross-browser on desktop devices	🟩	🟥	🟥

🟩 Yes 🟧 Partly 🟥 No

Form Validation

- It is common to use JavaScript to validate form information before submitting it to the web server
  - Is the name entered?
  - Is the e-mail address of correct format?
  - Is the phone number in the correct format?

Validating Form Fields

- Use the "" or null to check if a form field has information

```
if (document.forms[0].userName.value == "" ) {
       alert("Name field cannot be empty.");
       return false;
     } // end if
```

CHANGING STYLE FROM JAVASCRIPT

CSS attributes from Javascript

- How to set a CSS attributes on a html-node in the DOM?

  → You simply set it through the **style** attribute

- E.g. to set the width of an element:

```
var obj = document.getElementById("myId");
obj.style.width = "100px";
```

- In JavaScript, document.getElementById("myId") is similar in function to the CSS selector # myId.

CSS Properties with Hyphens

- JavaScript does not allow hyphens in names, so "camelCase" is used instead
- E.g.: the CSS property **`background-color`** is accessed by **`backgroundColor`** from JavaScript

```
var obj = document.getElementById("myId");
obj.style.backgroundColor = "#ff0000";
```

Unofficial Attributes

- Unofficial style attributes like `-webkit-background-size` can be set with this syntax:

```
var obj = document.getElementById("myId");
obj.style["-webkit-background-size"] = "400px"
```

Separate Behaviour and Style

- The previous examples mix style and behaviour☹
  - This makes the site/app more difficult to maintain, as the designer would have to work with both the CSS and the JavaScript files

- A cleaner approach is to define all the different styles in the CSS file – e.g. as different classes, and then change the element’s class name from JavaScript:

```
var obj = document.getElementById("myId");
obj.className = "newClass";
```

✓

class in HTML becomes className in JavaScript

web workers

- JavaScript running in the browser is mainly a single threaded environment (ES5)
- But JavaScript has some multithreading capabilities
  - The xmlHttpRequest object can be used to send asynchronous requests to a web server (ajax)
  - Web workers are used to spawn a new thread

- A web worker is an isolated JavaScript environment that runs alongside the main program for a document
  - Web workers and the main program interact via message passing using the postMessage() method and the message event

- For more detailed info: https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/Using_web_workers

Web Worker Example

Main program

```
var squareWorker = new Worker("code/squareworker.js");

squareWorker.addEventListener("message", function(event) {
  console.log("The worker responded:", event.data);
});

squareWorker.postMessage(10);
squareWorker.postMessage(24);
```

code/squareworker.js

```
addEventListener("message", function(event) {
  postMessage(event.data * event.data);
});
```

JavaScript Debugging

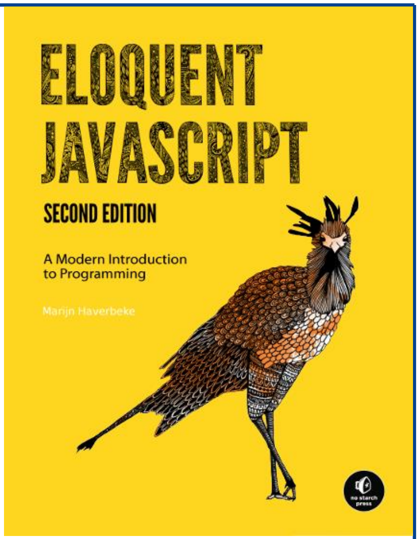
- Check the syntax of the statements
  - Pay very close attention to upper and lower case letters, spaces, and quotations
- Verify that you have saved the page with your most recent changes
- Verify that you are testing the most recent version of the page (**refresh or reload the page**)
- If you get an error message, use the error messages that are displayed by the browser (press F12 – debug mode)
  - In Firefox: Select Tools > Error Console
- Use the "use strict"; directive
  http://www.w3schools.com/js/js_strict.asp
- **Use JsLint**
  http://www.jslint.com/

How do I unit test JavaScript Code?

- **Mocha** (with **Chai** as assertion library) and **Jasmine** are two popular testing frameworks for JavaScript
- **Karma** is a test runner that can run both Jasmine and Mocha-style tests

- **Selenium** is a web driver that is often used for integration test
  - Using a browser to actually render and load the page, simulating user interactions, and checking the result

References and Links



- **Eloquent JavaScript** by Marijn Haverbeke
  http://eloquentjavascript.net
- Wikipedia has a superb overview
  http://en.wikipedia.org/wiki/DOM_events
- JavaScript libraries and frameworks overview
  http://www.javascriptoo.com/
- Microjs
  http://microjs.com/#