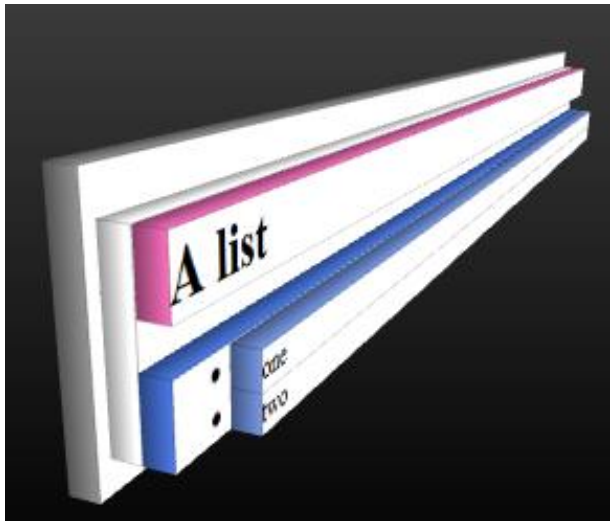
The University Of Sheffield.

COM1008: Web and Internet Technology

Lecture 11: The DOM



Dr. Steve Maddock
s.maddock@sheffield.ac.uk

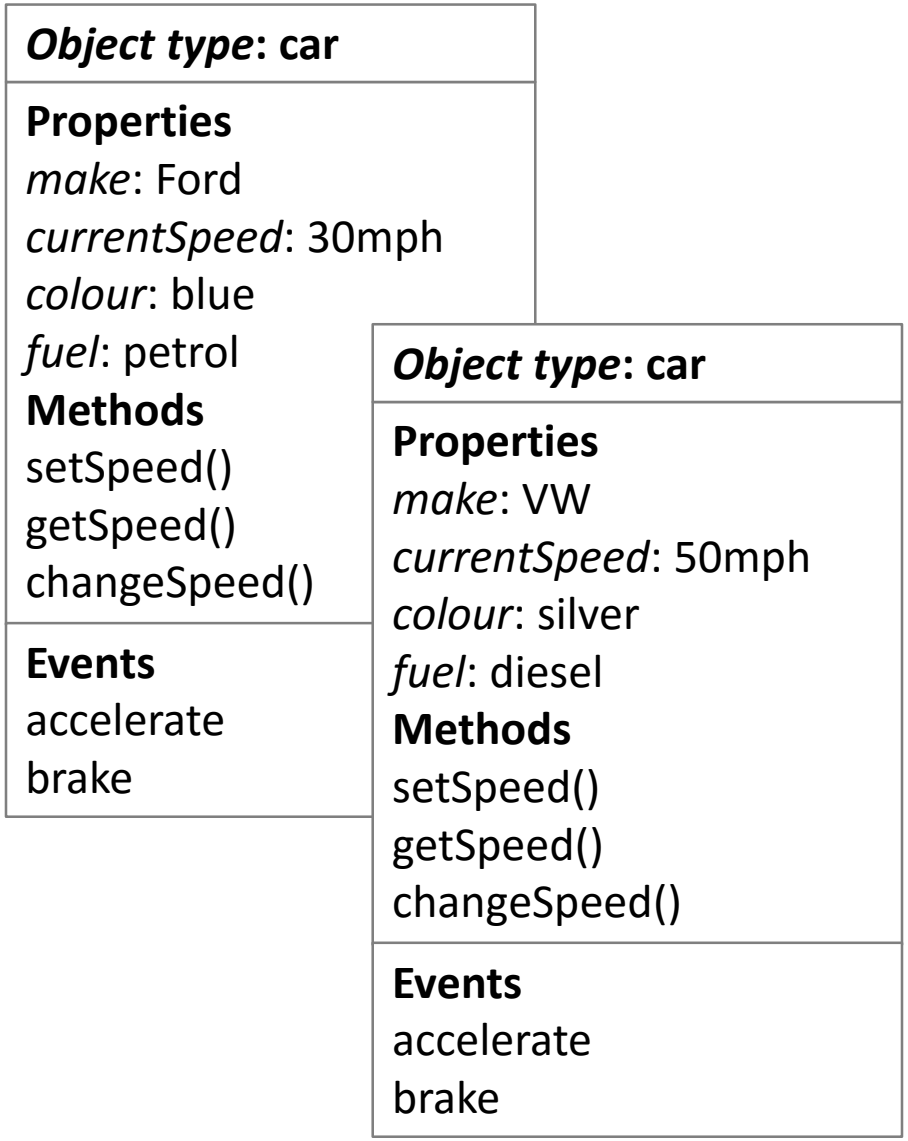
1. Introduction

- Structure and appearance using HTML and CSS
- Behaviour using JavaScript
  - *So far*: get input, represent data and do calculations, write output
- In fact, the Web browser supports more advanced features:
  - A programming environment to create and delete and manipulate elements of the Web page
  - An event-driven programming model to respond to user input
- Today we'll focus on the **Document Object Model**

- *References*
  - https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model
  - Jon Duckett, JavaScript & jQuery: interactive front-end web development, Wiley, 2014 (http://www.javascriptbook.com/)

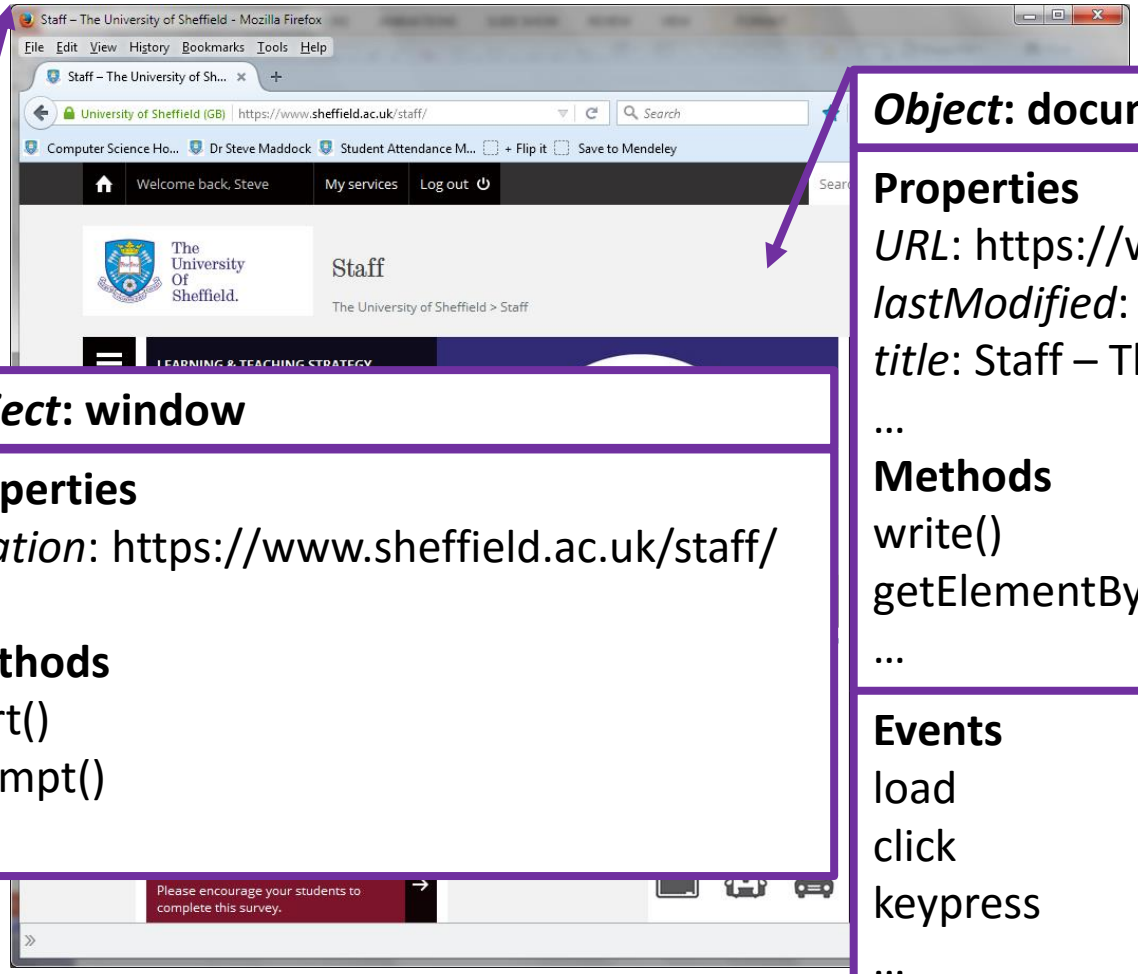
2. Objects

- In computer programming, an **object** can be used to represent a physical thing
- *Example*: A car
- We create instances of the car object
- Properties are name-value pairs
- Methods can be used to set, query and change the properties
- Events are interactions with objects



2.1 Objects created by the Web browser

- When you load a Web page, two objects are automatically created



Object: window

Properties

location: https://www.sheffield.ac.uk/staff/

…

Methods

alert()

prompt()

…

Object: document

Properties

URL: https://www.sheffield.ac.uk/staff/

lastModified: 07/11/2015 14:10:42

title: Staff – The University of Sheffield

…

Methods

write()

getElementById()

…

Events

load

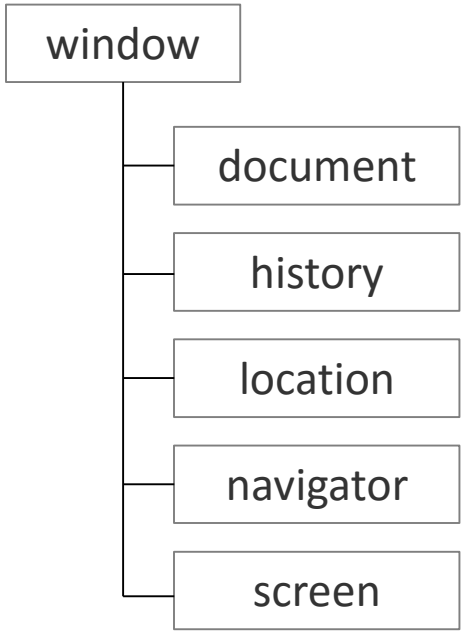
click

keypress

…

2.2 Three groups of built-in objects

- Browser Object Model
  - Objects that represent the window or the tab in the window
  - Also includes an object for the browser history
- Document Object Model
  - Objects that represent the page being displayed
  - Objects for each element and each section of text
- Global JavaScript Objects
  - Examples: Date, String, Array, …



Example

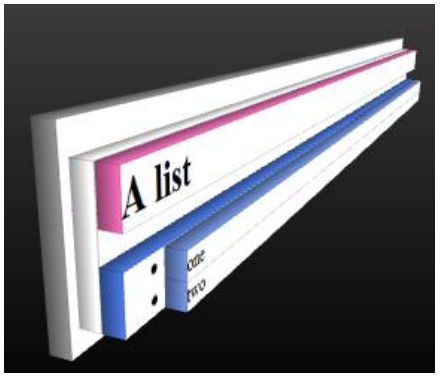
```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>JavaScript examples</title>
</head>

<body>

  <h1>Example</h1>
  <script>
    var length = prompt("Rectangle length in cm?");
    var width = prompt("Rectangle width in cm?");

    document.write("Area = " + length*width);
    alert("Area = " + length*width);
  </script>

</body>

</html>
```

In JavaScript, use the '.' operator to access an object's methods and properties.
Note: In Java, the properties are usually private.

The window object is different.
You can use alert(something);
rather than window.alert(something);

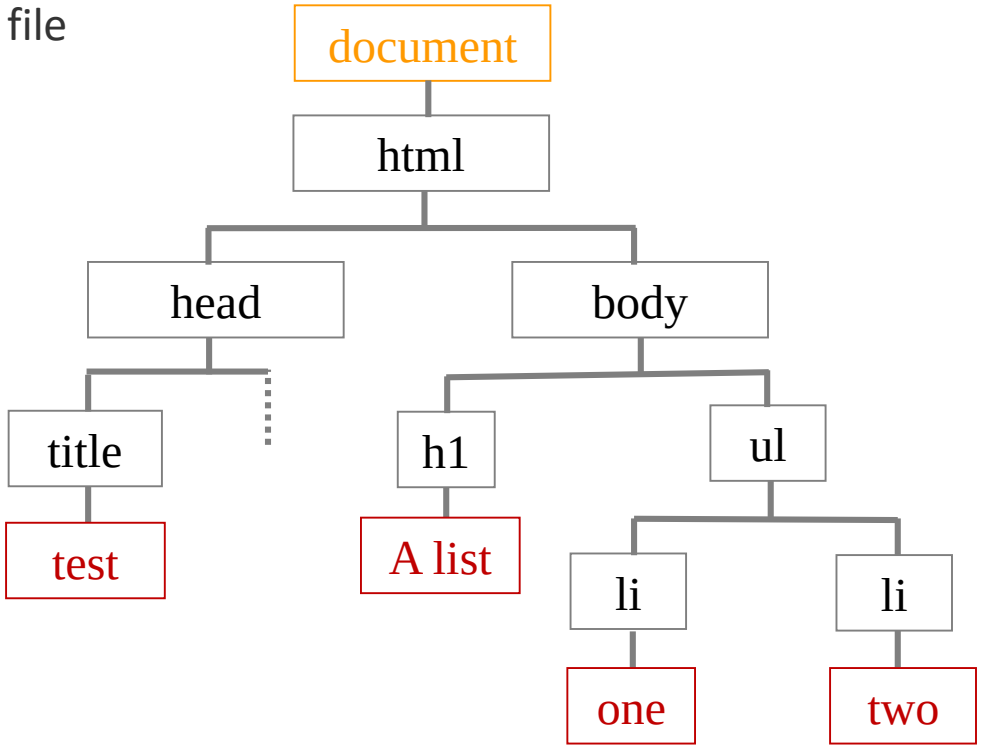
http://www.html5rocks.com/en/tutorials/internals/howbrowserswork/

3. How a browser sees a Web page



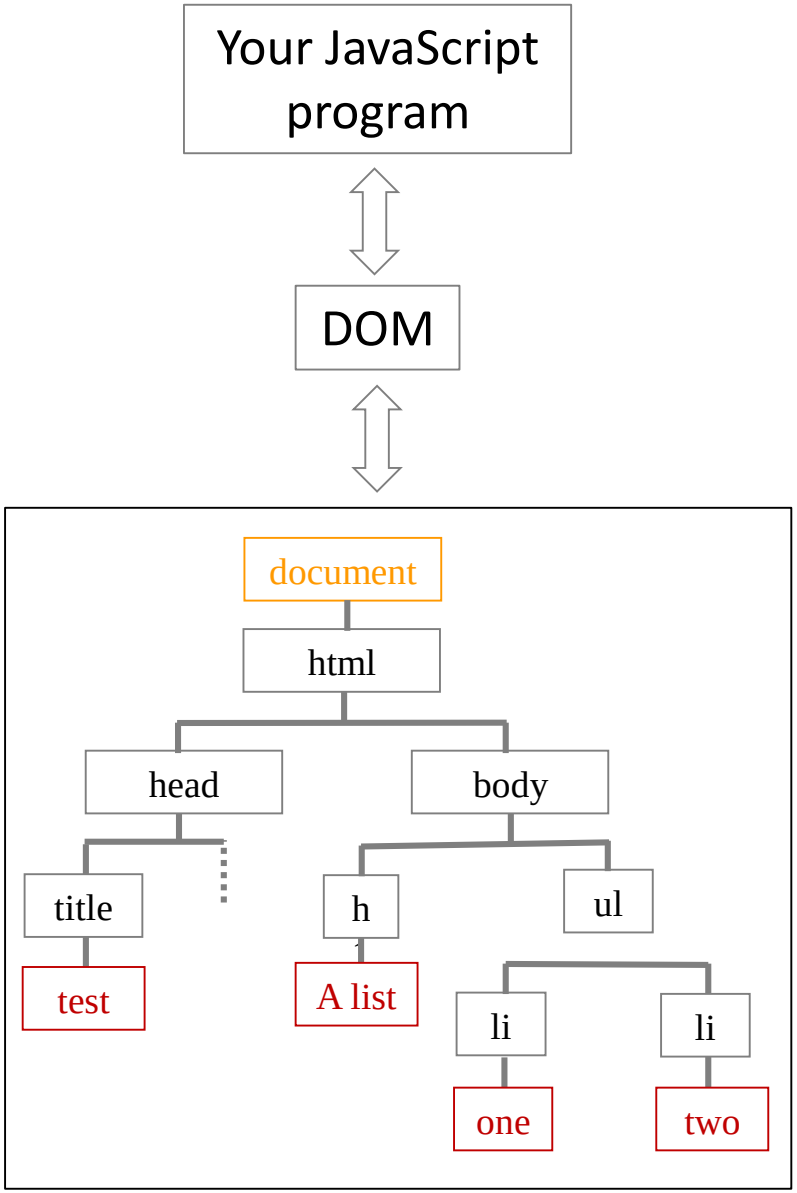
- The web browser receives a page as HTML
- It creates a model of the page and stores it in memory
  - Represented as a set of objects
- Uses a rendering engine to display the page
  - Use default style or linked CSS file

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>test</title>
</head>
<body>
  <h1>A list</h1>
  <ul>
    <li>one</li>
    <li>two</li>
  </ul>
</body>
</html>
```



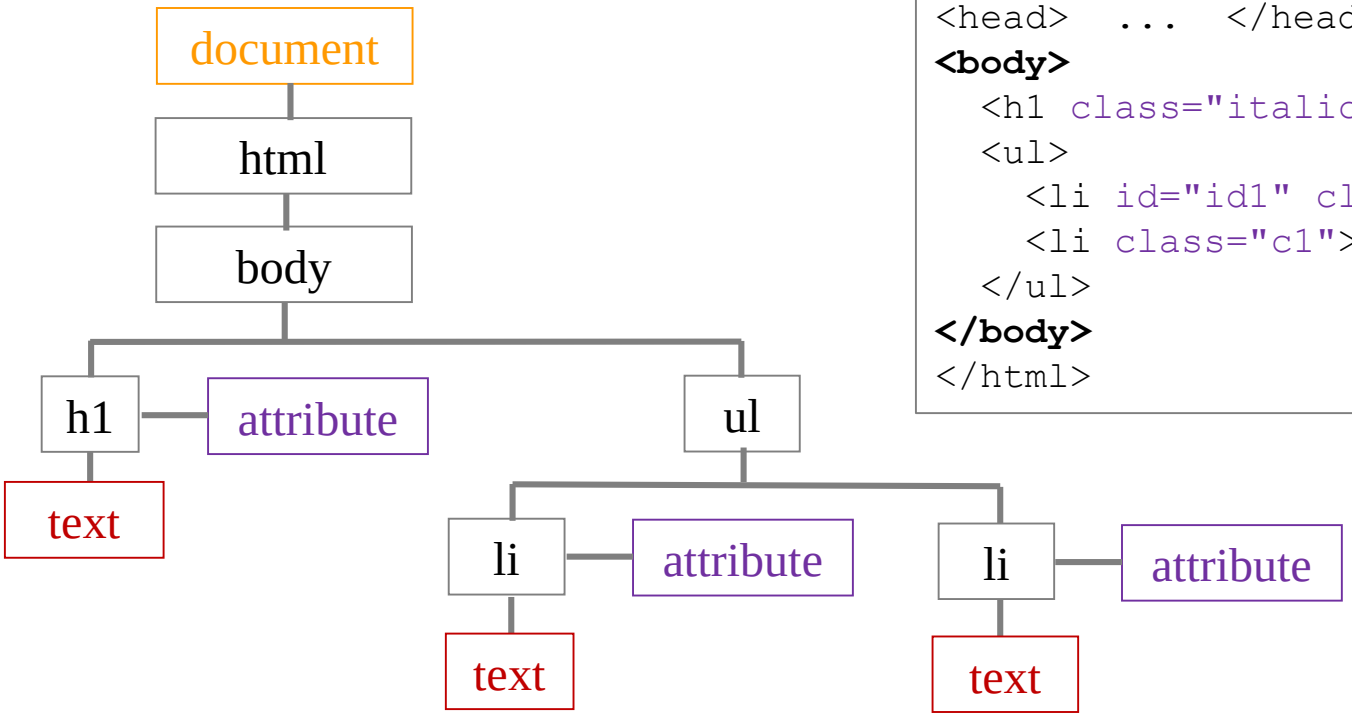
4. The DOM

- A set of rules implemented by browsers
- Represents a model of the HTML
  - A tree made of objects
- Methods and properties to manipulate the tree of objects
  - Examples: Add a new element, change the properties of an element, etc
- Often referred to as an Application Programming Interface (API)
  - Your JavaScript program uses the DOM to speak to the browser



4.1 DOM nodes

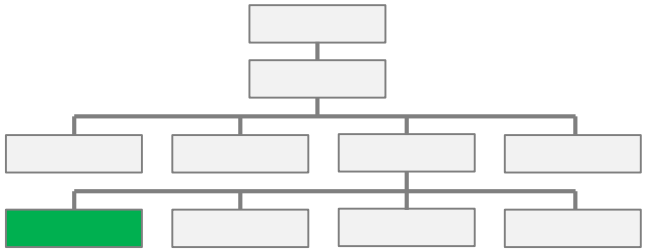
- Every element, attribute and piece of text is a DOM node
- Attributes are part of the element, not a child
- Text nodes cannot have children



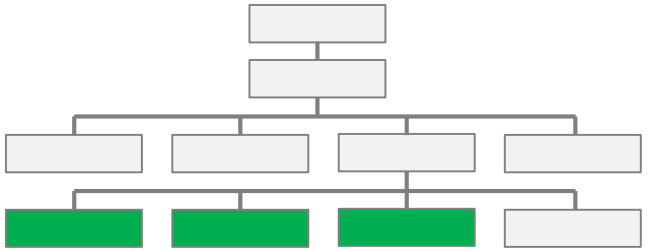

```
<!DOCTYPE html>
<html lang="en">
<head>  ...  </head>
<body>
  <h1 class="italic">A list</h1>
  <ul>
    <li id="id1" class="c1">one</li>
    <li class="c1">two</li>
  </ul>
</body>
</html>
```

4.2 Step 1: Access the elements

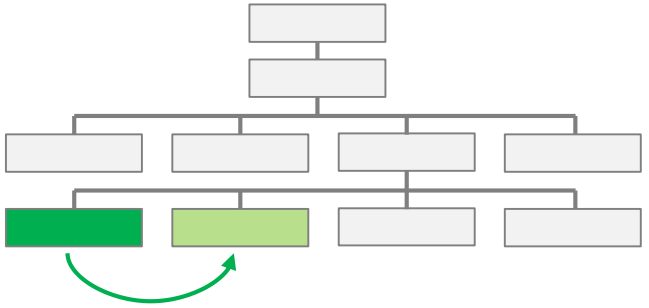
- Select an individual element node
  - getElementById(), querySelector(), …



- Select multiple elements (a NodeList)
  - A collection of nodes; access each using an index
  - getElementsByTagName(), getElementsByClassName(), querySelectorAll(), …
  - *Example*: Find all the h1 elements in the document

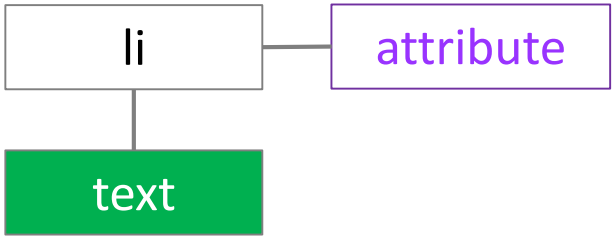


- Traverse between element nodes
  - parentNode, previousSibling, nextSibling, firstChild, lastChild, childNodes[i], …

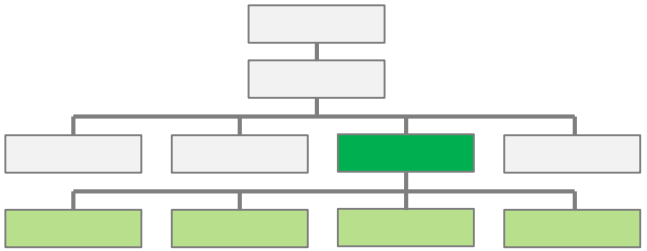


4.3 Step 2. Work with those elements

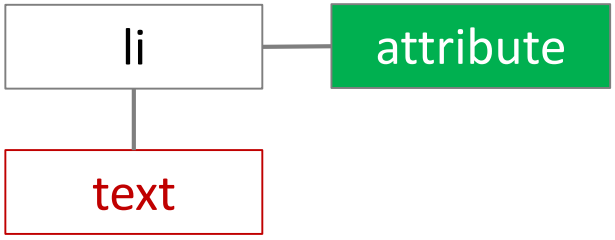
- Access / update text nodes
  - nodeValue …



- Work with HTML content
  - innerHTML, textContent, createElement(), createTextNode(), appendChild(), removeChild()…
  - *Examples*: Add a new list item to an existing list, create a new paragraph of text



- Access / update attribute values
  - className, id, hasAttribute(), getAttribute(), setAttribute(), removeAttribute(), …
  - *Example*: Make all h1 elements a different colour



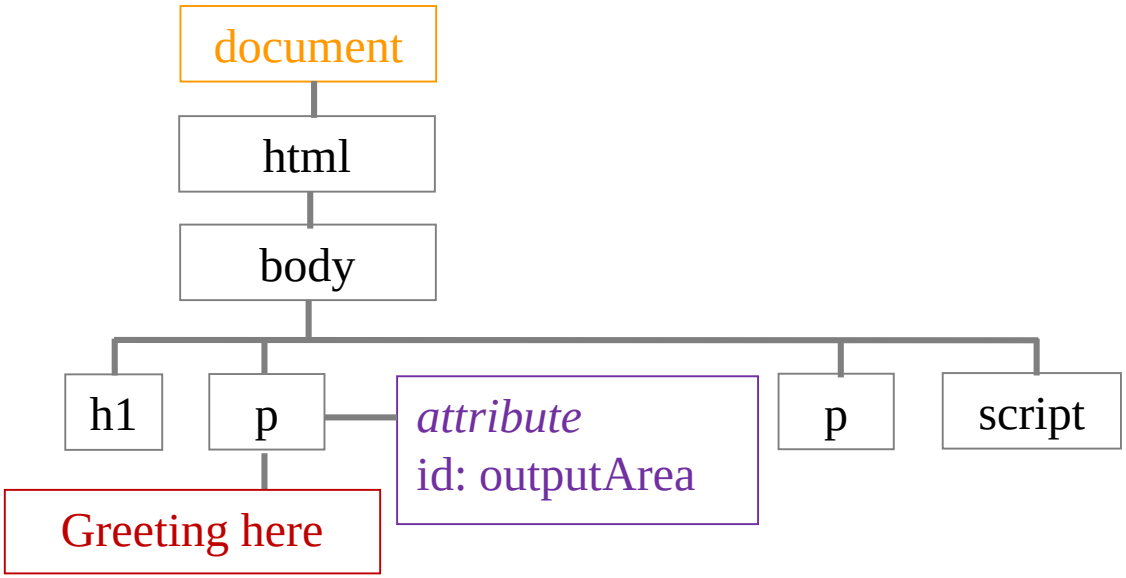
4.4 An example

```
<body>
  <h1>Example: Date</h1>
  <p id="outputArea">
  Greeting here
  </p>
  <p>Some more text</p>
  <script src="./js/date2.js">
  </script>
</body>
```

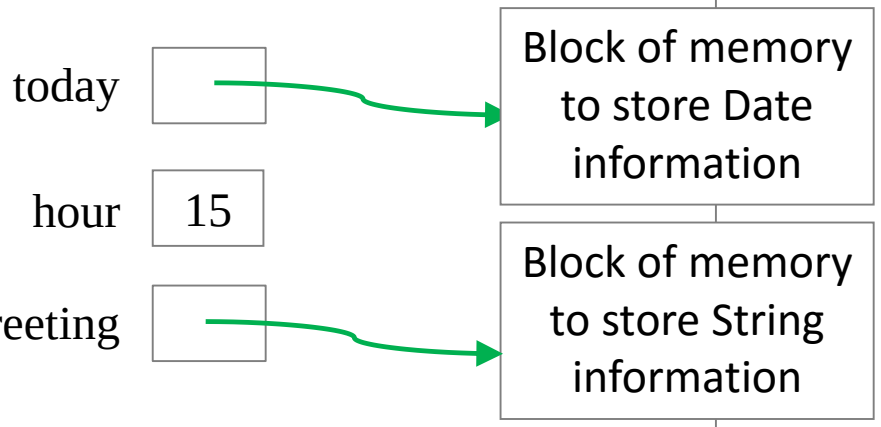
```
var today = new Date();
var hour = today.getHours();
var greeting = "";
if (hour < 12) {
  greeting = "Good morning";
}
else if (hour < 18) {
  greeting = "Good afternoon";
}
else {
  greeting = "Good evening";
}
var myElement = document.getElementById('outputArea');
myElement.textContent = greeting;
```

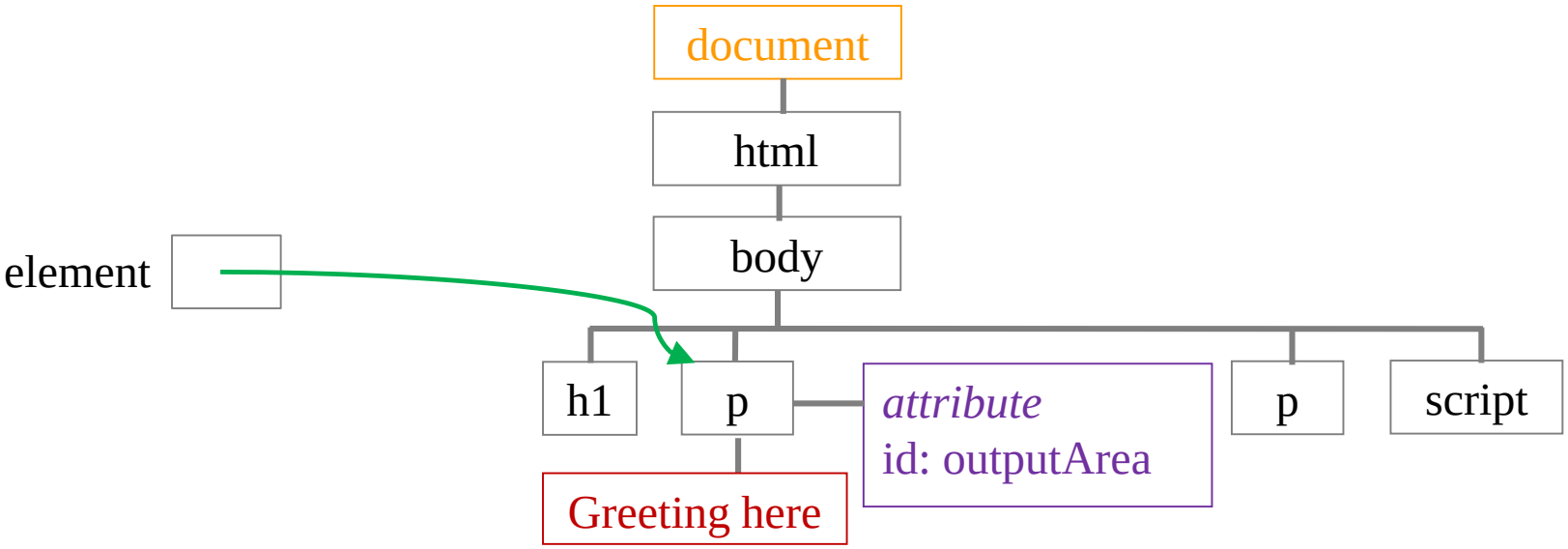
Don't use **element** as a variable name because it is a reserved word

```
<body>
  <h1>Example: Date</h1>
  <p id="outputArea">
  Greeting here
  </p>
  <p>Some more text</p>
  <script src="./js/date2.js">
  </script>
</body>
```

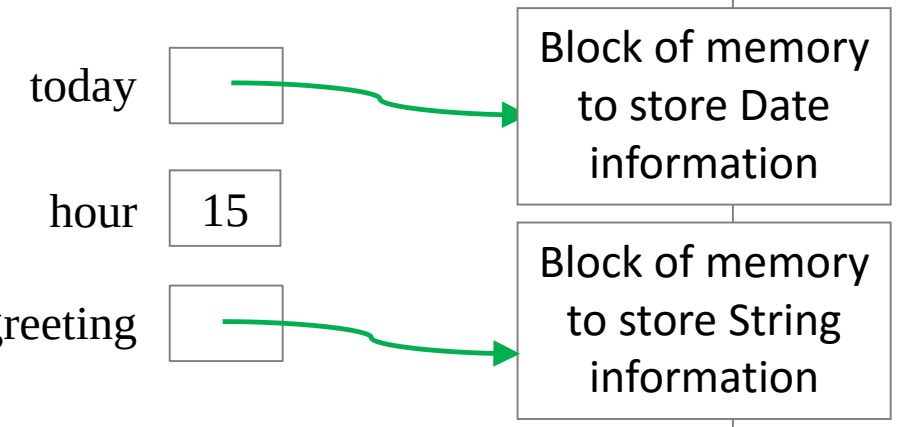


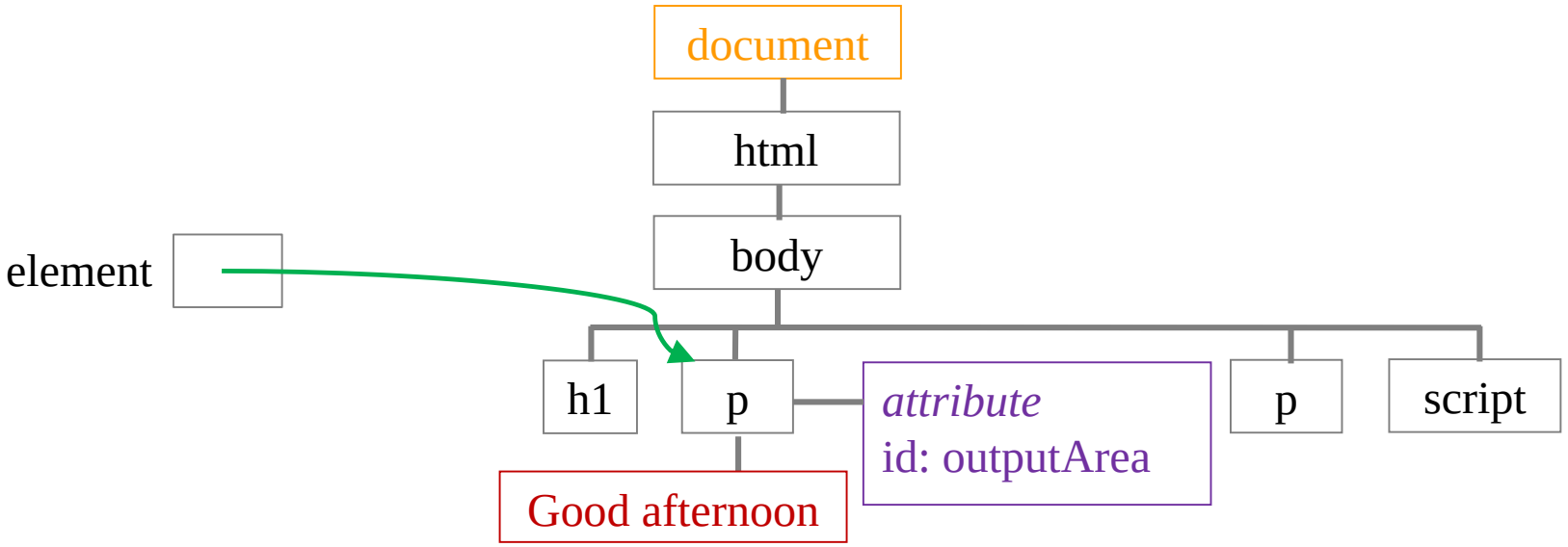
```
var today = new Date();
var hour = today.getHours();
var greeting = "";
if (hour < 12) {
  greeting = "Good morning";
}
else if (hour < 18) {
  greeting = "Good afternoon";
}
else {
  greeting = "Good evening";
}
var myElement = document.getElementById('outputArea');
myElement.textContent = greeting;
```





```
var today = new Date();
var hour = today.getHours();
var greeting = "";
if (hour < 12) {
  greeting = "Good morning";
}
else if (hour < 18) {
  greeting = "Good afternoon";
}
else {
  greeting = "Good evening";
}
var myElement = document.getElementById('outputArea');
myElement.textContent = greeting;
```





```
var today = new Date();
var hour = today.getHours();
var greeting = "";
if (hour < 12) {
  greeting = "Good morning";
}
else if (hour < 18) {
  greeting = "Good afternoon";
}
else {
  greeting = "Good evening";
}
var myElement = document.getElementById('outputArea');
myElement.textContent = greeting;
console.log(myElement);
```

today

hour 15

greeting

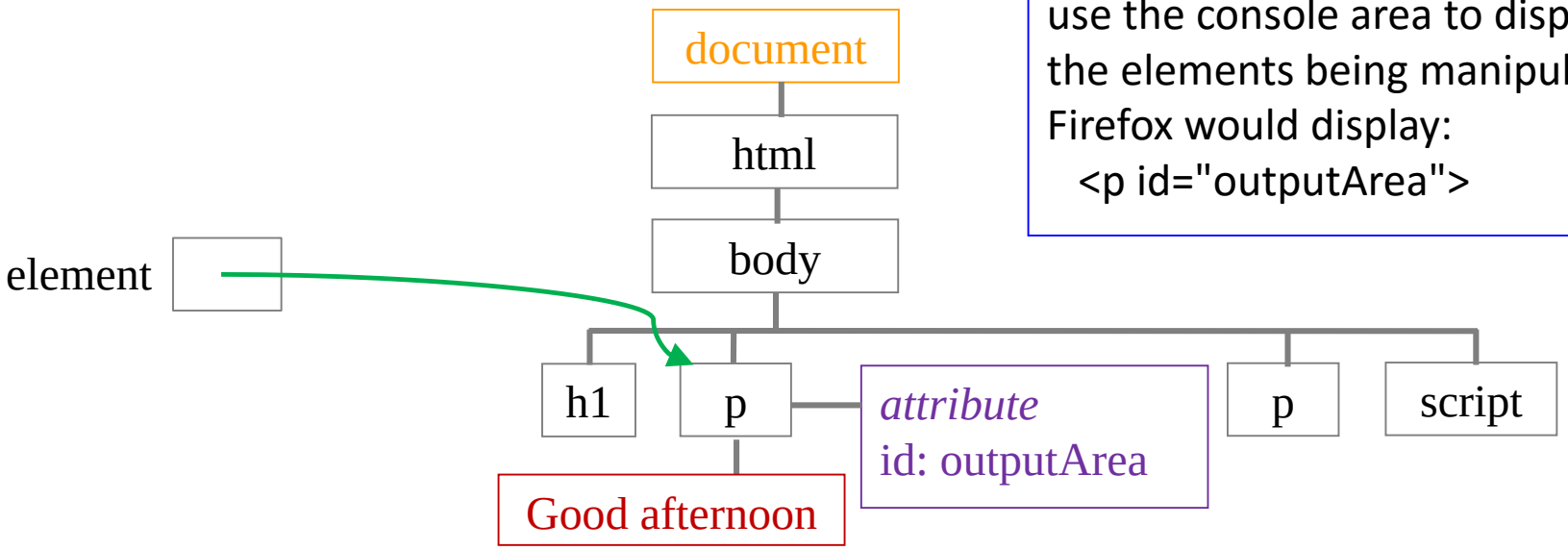
Block of memory to store Date information

Block of memory to store String information

DEBUGGING

Sometimes, it can be useful to use the console area to display the elements being manipulated. Firefox would display:

<p id="outputArea">

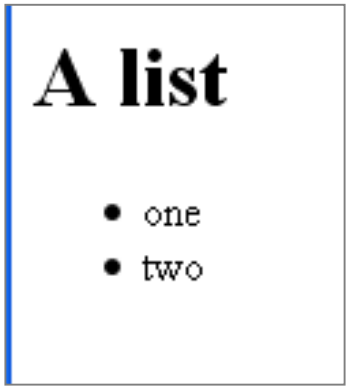


5. More on the DOM tree

- **element nodes**
  - h1, p, …
- **text nodes**
  - ‘A list’, ‘one’, etc
- *Example:* **text[one]** is a child of an **li** element which is a child of a **ul** element, etc.
- The **new line characters** can cause problems as they are interpreted as extra **text nodes**

```
document
  html
    head
      sub-tree for head contents
    body
      text[\n]
      h1
        text[A list]
      text[\n]
      ul
        text[\n]
        li
          text[one]
        text[\n]
        li
          text[two]
        text[\n]
      text[\n]
```

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>test</title>
</head>
<body>
  <h1>A list</h1>
  <ul>
    <li>one</li>
    <li>two</li>
  </ul>
</body>
</html>
```



6. getElementsByTagName('tag')

demo

gatetag1.html

```
<body>
  <ul id="mylist">
    <li>one</li>
    <li>two</li>
    <li>three</li>
  </ul>
  <script src="./js/gettag1.js"></script>
</body>
```

gettag1.js

```
var listItems = document.getElementsByTagName('li');
console.log(listItems);
for (var i=0; i<listItems.length; i++) {
  console.log(listItems[i]);
}
```

console

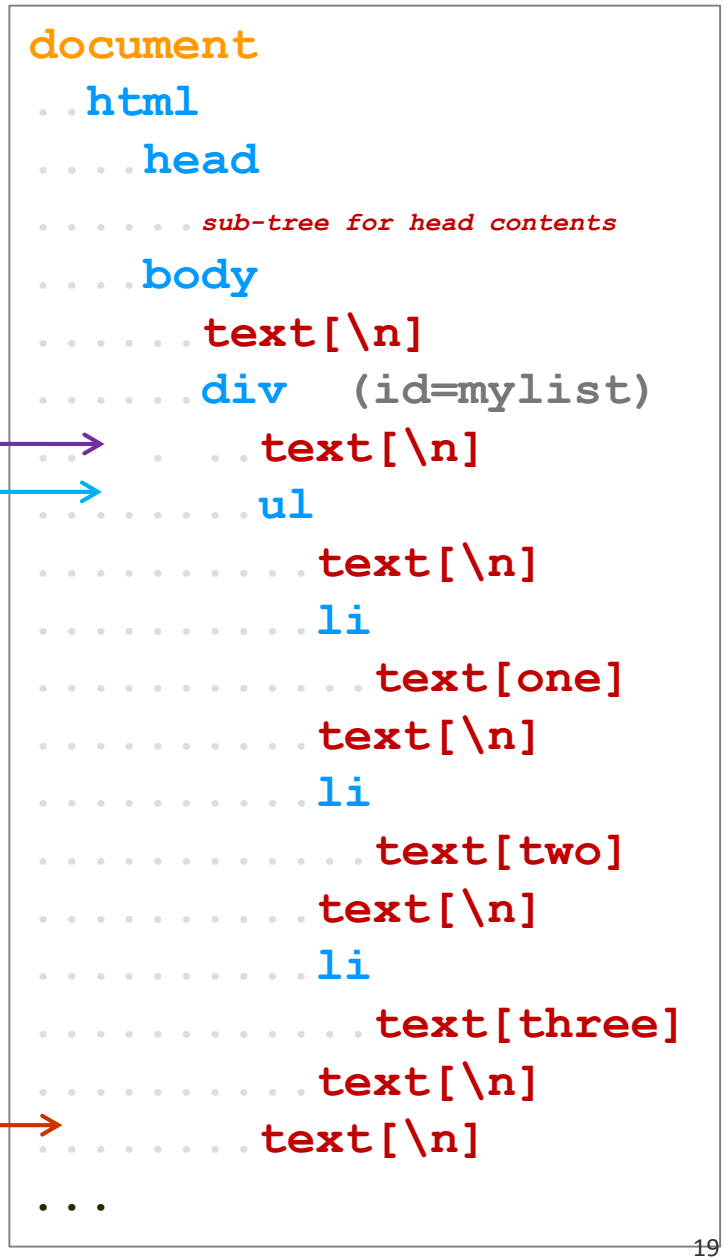
```
HTMLCollection [ <li>, <li>, <li> ]
<li>
<li>
<li>
```

- returns all elements with the name *tag* as a NodeList, which is a list of elements
- Here, all the li tags are returned

7. childNodes

- In this example, the div element has three children:
  - A text node (end of line character)
  - A ul element node
  - A text node (end of line character)

```
<body>
  <div id="mylist">
    <ul>
      <li>one</li>
      <li>two</li>
      <li>three</li>
    </ul>
  </div>
</body>
```



demo

7. childNodes

childnodes.html

```
<body>
  <div id="mylist">
    <ul>
      <li>one</li>
      <li>two</li>
      <li>three</li>
    </ul>
  </div>
  <script src="./js/childnodes.js"></script>
</body>
```

childnodes.js

```
var mylist = document.getElementById("mylist");
console.log(mylist);
console.log(mylist.firstChild);
console.log(mylist.childNodes[0]);
var nodes = mylist.childNodes;
console.log(nodes);
console.log(nodes.item(0));
console.log(nodes.item(1));
console.log(nodes[2]);
```

- A list of all the nodes inside the current one
- Use array notation or method ‘item’ to access each child node

7. childNodes demo

```
<body>
  <div id="mylist">
    <ul>
      <li>one</li>
      <li>two</li>
      <li>three</li>
    </ul>
  </div>
  <script src="./js/childnodes.js"></scrip
</body>
```

console

```
<div id="mylist">
#text "
"
#text "
"
NodeList [ #text "
", <ul>, #text "
" ]
#text "
"
<ul>
#text "
"
```

```
var mylist = document.getElementById("mylist");
console.log(mylist);
console.log(mylist.firstChild);
console.log(mylist.childNodes[0]);
var nodes = mylist.childNodes;
console.log(nodes);
console.log(nodes.item(0));
console.log(nodes.item(1));
console.log(nodes[2]);
```

8. Examples of using the DOM

returns the fourth link inside the element with the ID 'navigation'

```
document.getElementById('navigation').getElementsByTagName('a')[3];
```

returns the fourth link inside the element with the ID 'navigation'

```
var e = document.getElementById('navigation');
var fourthLink = e.getElementsByTagName('a')[3];
```

returns the first paragraph inside the third div in the document

```
document.getElementsByTagName('div')[2].getElementsByTagName('p')[0];
```

returns the 4th element's first sub element inside the element with the ID nav

```
var other = document.getElementById('nav').childNodes[3].firstChild;
```

returns the third node inside the previous element that is on the same level as the parent element of o

```
var prevlink = o.parentNode.previousSibling.firstChild.childnodes[2];
```

9. Creating new content

- createElement(element) – creates a new element
- createTextNode(string) – creates a new text node with the value string

```
<body>

  <h1>Results</h1>
  <div id="results">
    <p>New text will be added after this...</p>
  </div>

  <h1>Some other heading</h1>
  <p>Some other paragraph</p>

  <script src="./js/results_button.js"></scri

</body>
```

Insert new text here using JavaScript

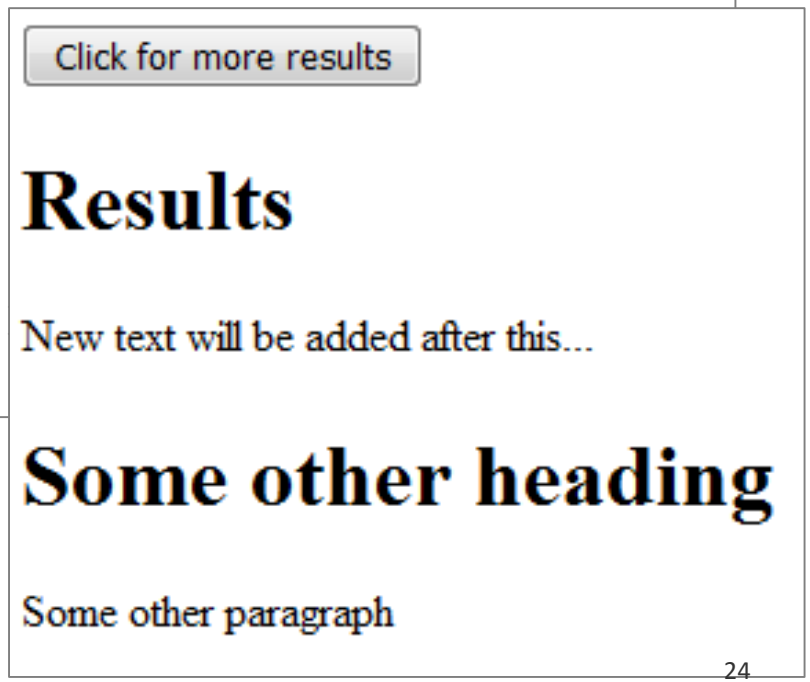
Plan:

- Access the div element
- Create a new paragraph element
- Append it as a child of the div element

9. Creating new content

demo

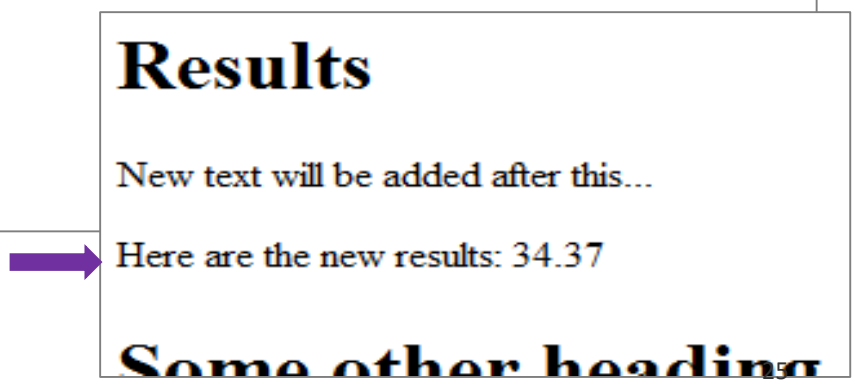
```
<body>
  <button name="moreResults" id="moreResults">
    Click for more results
  </button>

  <h1>Results</h1>
  <div id="results">
    <p>New text will be added after this...</p>
  </div>

  <h1>Some other heading</h1>
  <p>Some other paragraph</p>

  <script src="./js/results_button.
</body>
```



9. Creating new content

demo

```
function updateResults() {
  var results = document.getElementById("results");
  var myElement = document.createElement("p");
  var n = Math.random()*50;
  var str = "Here are the new results: " + n.toFixed(2);
  var textNode = document.createTextNode(str);
  myElement.appendChild(textNode);
  results.appendChild(myElement);
}

function addButtonhandler() {
  var b = document.getElementById("moreResults");
  b.addEventListener('click', updateResults, false);
}

// main program
addButtonhandler();
```



10. Where does the JavaScript program go?

- The page has to be loaded before the JavaScript program runs otherwise there is no DOM tree to work with

Solution 1. Put the script just before the close body tag

```
<!DOCTYPE html>
<html lang="en">
<head>
  ...
</head>
<body>
  ...
  <script src="file.js">
  </script>
</body>
</html>
```

Solution 2. Put the script in the head tag and find a way to run it **after** the page has loaded

```
<!DOCTYPE html>
<html lang="en">
<head>
  ...
  <script src="file.js">
  </script>
</head>
<body>
  ...
</body>
</html>
```

10.1 *Solution 1.* Script just before the close body tag

- Advantage: page is already loaded before script is run
- Advantage: does not slow down loading of page

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>JavaScript examples</title>
</head>
<body>
  <h1>Example: Date</h1>
  <p id="outputArea">Greeting here</p>
  <p>Some more text</p>
  <script src="./js/date2.js"></script>
</body>
</html>
```

date2.js

```
var today = new Date();
console.log("Date=" + today);

var hour = today.getHours();
console.log("The hour is: " + hour);

var greeting = "";
if (hour < 12) {
  greeting = "Good morning"
}
else if (hour < 18) {
  greeting = "Good afternoon"
}
else {
  greeting = "Good evening"
}

var myElement = document.getElementById('outputArea');
myElement.textContent = greeting;
```

10.2 Solution 2

- *Solution 2*. Put the script in the head tag and find a way to run it after the page has loaded
- *Disadvantage*: pages can seem slower to load

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>JavaScript examples</title>
  <!-- How to include script here and run it when the page has loaded? -->
</head>
<body>
  <h1>Example: Date</h1>
  <p id="outputArea">Greeting here</p>
  <p>Some more text</p>
</body>
</html>
```

11.1 Event listener

```
function getGreeting() {
  var today = new Date();          console.log("Date=" + today);
  var hour = today.getHours();     console.log("The hour is: " + hour);
  var greeting = "";
  if (hour < 12) { greeting = "Good morning" }
  else if (hour < 18) { greeting = "Good afternoon" }
  else { greeting = "Good evening" }
  return greeting;
}

function ready() {
  var myElement = document.getElementById('outputArea');
  myElement.textContent = getGreeting();
  console.log(myElement);
}


// main program
window.addEventListener('load', ready, false);
```

The code is wrapped in a function(s) and called when the event is fired

If the script is in the head tag, use an event listener to run it when the page has finished loading

Warning: addEventListener() method is not supported in Internet Explorer 8 and earlier versions

11.1 Event listener

```
function getGreeting() {
  var today = new Date();          console.log("Date=" + today);
  var hour = today.getHours();     console.log("The hour is: " + hour);
  var greeting = "";
  if (hour < 12) { greeting = "Good morning" }
  else if (hour < 18) { greeting = "Good afternoon" }
  else { greeting = "Good evening" }
  return greeting;
}

function ready() {
  var myElement = document.getElementById('outputArea');
  myElement.textContent = getGreeting();
  console.log(myElement);
}

// main program
if (window.addEventListener) {
  window.addEventListener('load', ready, false);
}
```

Object detection, to confirm that a method is actually available before it is called

12. Unobtrusive JavaScript

- “To separate JavaScript from HTML markup, as well as keeping modules of JavaScript independent of other modules”
  - Do not add JavaScript directly to the document
  - Include, by using `<script src="script.js"></script>`
- “Unobtrusive JavaScript should degrade gracefully - all content should be available without all or any of the JavaScript running successfully”
- “Unobtrusive JavaScript should not degrade the accessibility of the HTML, and ideally should improve it, whether the user has personal disabilities or are using an unusual, or unusually configured, browser”

(Flanagan, 2006)

Flanagan, David (2006). JavaScript: The Definitive Guide (5th ed.). O'Reilly & Associates. p. 241

13. JavaScript libraries

- Easier development of JavaScript-based applications
  - Deal with browser inconsistencies
- One of the most widely used is jQuery
  - Lots of functions: Add content to a page, replace and remove selections, reading and changing CSS properties, …
  - Free, Lightweight footprint, CSS3 compliant, Large developer community, Lots of plug-ins, Tried and tested – used by Google, Dell, Mozilla, Wordpress, …
  - Quick reference: http://oscarotero.com/jquery/
- Lots of other libraries:
  - DOM manipulation: Dojo, jQuery, MooTools, Prototype, YUI
  - Graphics and charts: Chart.js, D3.js, Processing.js, Raphael, Three.js
  - Web applications: AngularJS, Backbone.js, Ember.js
  - Other: Bootstrap, Modernizr

14. Summary

- There are three groups of built-in objects:
  - Browser Object Model, Document Object Model, Global JavaScript Objects
- The DOM is a set of rules implemented by browsers
  - Represents a model of the HTML – a tree made of objects
  - Methods and properties to manipulate the tree of objects
- The Web page has to be loaded before the JavaScript program runs otherwise there is no DOM tree to work with
- Aim: unobtrusive JavaScript
- There are many libraries to help with JavaScript programming

- *Next lecture:* Events and forms

Appendix A. innerHTML

- innerHTML can be used to retrieve and replace the content of a node (as noted in a previous lecture. Remember: beware XSS issues.)
- *Example*: add an extra child element to the body element

```
var table = document.createElement("table");
var tablecontents = "<tr><th>N</th><th>Stars</th></tr>";

for (var j=0; j<data.length; j++) {
  var n = Math.floor(data[j]);
  tablecontents += ("<tr><td>"+n+"</td><td>");
  for (var i=0; i<n; i++) {
    tablecontents += "*";
  }
  tablecontents += "</td></tr>";
}
table.innerHTML = tablecontents;
document.body.appendChild(table);
```

demo