DOM

Chapter 9

- Web browsers provide six important global objects that you can access in your JavaScript code.

Name	**Represents**
document	current web page
history	list of pages user has visited
location	url of current web page
navigator	web browser being used
screen	screen area occupied by browser and page
window	browser window

- **window** properties and events can be found at http://www.w3schools.com/jsref/obj_window.asp
- window events you should know include: alert, confirm, prompt, open, close, print, blur, focus, moveBy, moveTo, resizeBy, resizeTo, scrollBy, scrollTo.
- **document** methods and properties can be found at http://www.w3schools.com/jsref/dom_obj_document.asp
- document properties you should know include: body, cookie, referrer, title, URL.
- document methods you should know include: getElementById, getElementsByName.

- **navigator** properties and events can be found at http://www.w3schools.com/jsref/obj_navigator.asp
- navigator properties you should know include: appName, appVersion, cookieEnabled, language, platform, userAgent.
- **history** methods and properties can be found at http://www.w3schools.com/jsref/obj_history.asp
- history properties you should know include: length.
- history methods you should know include: back, forward, go.

- **screen** properties can be found at http://www.w3schools.com/jsref/obj_screen.asp
- screen properties you should know include: availHeight, availWidth, colorDepth, height, width, pixelDepth.
- **location** methods and properties can be found at http://www.w3schools.com/jsref/obj_location.asp
- location properties you should know include: hash, hostname, href, pathname, port, protocol, search.

- **Unobtrusive JavaScript** - a way of attaching event handlers without putting JavaScript into the body of the HTML code.
- *window* allows us to write unobtrusive JavaScript.
- Consider the example from the previous set of slides.
- This is poor separation of content and code. Ideally, all content is in HTML files, all presentation is in CSS files and all code is in js files.

```
<button onclick="compute();">Do Stuff!</button>
```

- DOM objects have properties matching HTML events. e.g. onclick or onchange.
- Assume our HTML code has the following:
  <button onclick="compute();">Do Stuff!</button>
- It would be better to assign that button an id:
  <button id="someID">Do Stuff!</button>
- Our js file would attach the event handler:

```
var someIDButton = document.getElementById("someID");
someIDButton.onclick = compute; //no parens needed
```

All JavaScript can be removed from our HTML file except the initial loading of the script:
<script src=. . .

- If one would do exactly as shown on the previous page, it wouldn't work.
- If the js code is put in the global section, an error would be reported.
- We must wait for the page's body to load. *window.onload* occurs when the browser has finished loading the page.

```
window.onload = startUp; //declared, not executed
...
function startUp(){
    var someIDButton = document.getElementById("someID");
    someIDButton.onclick = compute;
    //other handlers attached
}
```

- Some functions are only useful once and therefore don't really need a name.
- The *window.onload* function is one you will never call again.
- **Anonymous function** - a function that is not given a name and can only be called if it is stored in a variable.

```
window.onload = function(){
    var someIDButton = document.getElementById("someID");
    someIDButton.onclick = compute;
    //other handlers attached
}; //<--- don't forget the ending semicolon
```

- Unobtrusively attaching functions has a huge benefit in that the event handlers are bound to the element objects.
- The function can refer to the element object by using **this**.
- For global code or regular functions, *this* refers to the global window object.

Example

Write a tip calculator program to calculate 10%, 15% or 20%. Each calculation should be a button with a text box as the original total. The functions are almost identical with one small change. We can use **this** to shorten our code.

- Changing text inside an element should use *textContent*. However, IE doesn't support it. All browsers support the non-standard *innerHTML*.
- The keyword *value* is used to get the value of text boxes and other form elements.

Example

Write a web page to read in a bunch of text into some text area. Also provide a “find” textbox and a “replace” text box. Clicking a button should replace all instances of the find text with the replace text.

- All DOM elements have a **style** property. This property is an object.
- DOM style properties are almost identical to their CSS counterparts with a few exceptions.
    - background-color → backgroundColor
    - border-top-width → borderTopWidth
    - float → cssFloat
    - . . .
- All DOM style properties are stored as strings, “” by default.
- http://www.w3schools.com/jsref/dom_obj_style.asp has a complete list.

```
var someParagraph = document.getElementById("test");
alert(someParagraph.style.color); //alert current color
someParagraph.style.color = "red";
```

Example

Modify the previous example to add some text showing how many text replacements were made. Change the text and background color of that text. Every time a the find/replace button is clicked, this text should increase in size by 2pt.

- Adding a bunch of styling to our JavaScript code is not always the best idea.
- Content - HTML, Code - JavaScript, Styling - CSS.
- Adding *this.style.color* = *"red"* breaks this separation.
- Ideally, we should use JavaScript to modify the class or ID of an element only.
- However, be careful not to destroy other classes that were a part of that element. If other classes are needed, use the classList property.

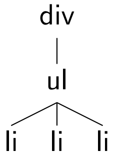
```
function someFunction(){
    //this.style.color = "red"; ☹
    this.className = "clickedbutton"; //☺
}
```

```
.clickedbutton{
    color: red;
}
```

- JavaScript represents each element on a web page as an object.
- These objects/nodes are connected to each other with parent, child and sibling links. These links make up the **DOM tree**.

Example

```
<div>
   <ul>
      <li>Content</li>
      <li>Content</li>
      <li>Content</li>
   </ul>
</div>
```



- **Element Node** - represents an HTML tag, can have children and attributes.
- **Text Node** - represents the inline text inside a block element. Always a child of an element node.
- **Attribute Node** - represents an attribute name/value pair inside an element's opening tag. It is not considered a child node; it is merely related to an element.
- We will mainly use element nodes but it's important to realize other nodes exist.

Example

<p>Here is some text with a <a href="someLink.php">link</a>!!!</p>

Every DOM object has the following properties. The property is either a node or null.

- **parentNode** - the element that contains the current node.
- **previousSibling** - previous node with the same parent.
- **nextSibling** - next node with the same parent.
- **firstChild** - first child of a node.
- **lastChild** - last child of a node.
- **childNodes** - an array of children.

Example

<p>Here is some text with a <a href="someLink.php">link</a>!!!</p>
This is how you would change link to cats:

```
para.firstChild.nextSibling.lastChild.textContent="cats";
//or you could do this
para.lastChild.previousSibling.lastChild.textContent="cats";
```

- Traversing the DOM tree as done on the previous page can be difficult.
- We generally want to modify a *group* of elements to have a new feature or style.
- **getElementById(id)** - returns an element with the given HTML id.
- **getElementsByTagName(tag)** - returns all elements with a given HTML tag, e.g. *p*.
- **getElementsByName(name)** - returns all elements with a given name attribute.

The following code will set all paragraphs on a page to have a yellow background:

```
var allParagraphs = document.getElementsByTagName("p");
for(var i = 0; i < allParagraphs.length; i++){
    allParagraphs[i].style.backgroundColor = "yellow";
}
```

A better way using unobtrusive JavaScript. In our CSS file:

```
.changeYellow {
    background-color: yellow;
}
```

```
var allParagraphs = document.getElementsByTagName("p");
for(var i = 0; i < allParagraphs.length; i++){
    allParagraphs[i].className = "changeYellow";
}
```

Selection methods exist for all DOM element objects:

.changeYellow {
  background-color: yellow;
}

```
var myFooter = document.getElementsById("footer");
var footPara = myFooter.getElementsByTagName("p");
for(var i = 0; i < footPara.length; i++){
    footPara[i].className = "changeYellow";
}
```

Example

Create a page that allows us to search for a particular word or phrase. Paragraphs should be highlights in yellow if they contain the word or phrase we are searching for.

Example

Repeat the previous example except only highlight the word or phrase being searched for. The code for this will likely be hackish. Don't worry about removing old highlighting, we will fix this later.

- We often want to modify all nodes with a given tag or CSS class or all elements in some list.
- **querySelector(selector)** - returns the first element that matches the given CSS selector.
- **querySelectorAll(selector)** - returns all elements that match the given CSS selector.
- The following selector: "div#someID p.someClass" returns all paragraphs with a className of someClass contained inside of a div with an id of someID.

Example

Repeat the example from the previous slide except only highlight the word or phrase being searched for. Be sure to remove all old highlighting and only highlight words in the "text area" above (i.e. inside some div).

- It may be useful to add and remove elements from a page.
- **document.createElement("type")** - a method to create and return a new empty DOM node of type.
- A node won't appear until it is part of the pages DOM tree. It must be added somewhere.
  - **appendChild(node)** - appends node to end of child list.
  - **insertBefore(new, old)** - inserts new node directly before old node in child list.
  - **removeChild(node)** - removes node from child list.
  - **replaceChild(new, old)** - replaces old with new.

```
var newPara = document.createElement("p");
newPara.className = "myFavClass";
newPara.innerHTML = "This is my new paragraph!";
var myDiv = document.getElementById("theDiv");
//add newPara to end of div's child list
myDiv.appendChild(newPara);
```