Q: How do you annoy a web developer?
The DOM tree

- The elements of a page are nested into a tree-like structure of objects the DOM has properties and methods for traversing this tree
DOM versus innerHTML hacking

Why not just code this way?

```javascript
function slideClick() {
    document.getElementById("main").innerHTML += "<p>A paragraph!</p>";
}
```

- Imagine that the new node is more complex:
  - ugly: bad style on many levels (e.g. JS code embedded within HTML)
  - error-prone: must carefully distinguish " and '
  - can only add at beginning or end, not in middle of child list

```javascript
function slideClick() {
    document.getElementById("main").innerHTML += "<p style='color: red; " + "margin-left: 50px;' " + "onclick='myOnClick();'>" + "A paragraph!</p>";
}
```
Creating new nodes

<table>
<thead>
<tr>
<th>name</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>document.createElement(&quot;tag&quot;)</code></td>
<td>creates and returns a new empty DOM node representing an element of that type</td>
</tr>
<tr>
<td><code>document.createTextNode(&quot;text&quot;)</code></td>
<td>creates and returns a text node containing given text</td>
</tr>
</tbody>
</table>

```js
// create a new <h2> node
var newHeading = document.createElement("h2");
newHeading.innerHTML = "This is a heading";
newHeading.style.color = "green";
```

- merely creating a element does not add it to the page
- you must add the new element as a child of an existing element on the page...
Modifying the DOM tree

Every DOM element object has these methods:

<table>
<thead>
<tr>
<th>name</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>appendChild(node)</code></td>
<td>places given node at end of this node's child list</td>
</tr>
<tr>
<td><code>insertBefore(new, old)</code></td>
<td>places the given new node in this node's child list just before old child</td>
</tr>
<tr>
<td><code>removeChild(node)</code></td>
<td>removes given node from this node's child list</td>
</tr>
<tr>
<td><code>replaceChild(new, old)</code></td>
<td>replaces given child with new node</td>
</tr>
</tbody>
</table>

```javascript
var p = document.createElement("p");
p.innerHTML = "A paragraph!";
document.getElementById("main").appendChild(p);
```

A paragraph!
Complex DOM manipulation problems

How would we do each of the following in JavaScript code? Each involves modifying each one of a group of elements ...

- When the Go button is clicked, reposition all the divs of class puzzle to random x/y locations.
- When the user hovers over the maze boundary, turn all maze walls red.
- Change every other item in the ul list with id of TAs to have a gray background.
Selecting groups of DOM objects

- methods in document and other DOM objects (* = HTML5):

<table>
<thead>
<tr>
<th>name</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getElementsByTagName</td>
<td>returns array of descendents with the given tag, such as &quot;div&quot;</td>
</tr>
<tr>
<td>getElementsByName</td>
<td>returns array of descendents with the given name attribute (mostly useful for accessing form controls)</td>
</tr>
<tr>
<td>querySelector *</td>
<td>returns the first element that would be matched by the given CSS selector string</td>
</tr>
<tr>
<td>querySelectorAll *</td>
<td>returns an array of all elements that would be matched by the given CSS selector string</td>
</tr>
</tbody>
</table>
Getting all elements of a certain type

highlight all paragraphs in the document:

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

```html
<body>
    <p>This is the first paragraph</p>
    <p>This is the second paragraph</p>
    <p>You get the idea...</p>
</body>
```
Complex selectors

highlight all paragraphs inside of the section with ID "address":

```javascript
// document.getElementById("address").getElementsByTagName("p")
var addrParas = document.querySelectorAll("#address p");
for (var i = 0; i < addrParas.length; i++) {
    addrParas[i].style.backgroundColor = "yellow";
}
```

```html
<p>This won't be returned!</p>
<div id="address">
    <p>1234 Street</p>
    <p>Atlanta, GA</p>
</div>
```
Common querySelectorAll issues

- many students forget to write . or # in front of a class or id

```javascript
// get all buttons with a class of "control"
var gameButtons = document.querySelectorAll("control");
var gameButtons = document.querySelectorAll(".control");
```

- querySelectorAll returns an array, not a single element; must loop over the results
  (document.querySelector returns just the first element that matches, if that's what you want)

```javascript
// set all buttons with a class of "control" to have red text
document.querySelectorAll(".gamebutton").style.color = "red";
var gameButtons = document.querySelector(".gamebutton");
for (var i = 0; i < gameButtons.length; i++) {
    gameButtons[i].style.color = "red";
}
```

Q: Can I still select a group of elements using querySelectorAll even if my CSS file doesn't have any style rule for that same group? (A: Yes!)
Problems with reading/changing styles

- **style** property lets you set any CSS style for an element
- problem: you cannot read existing styles with it
  
  *(you can read ones you set using the DOM .style, but not ones that are set in the CSS file)*
Accessing elements' existing styles

- getComputedStyle method of global window object accesses existing styles

```javascript
window.getComputedStyle(element).propertyName

function biggerFont() {
    // turn text yellow and make it bigger
    var clickMe = document.getElementById("clickme");
    var size = parseInt(window.getComputedStyle(clickMe).fontSize);
    clickMe.style.fontSize = (size + 4) + "pt";
}
```

Output

- Click Me
Common bug: incorrect usage of existing styles

• the following example computes e.g. "200px" + 100 + "px", which would evaluate to "200px100px"

```javascript
var main = document.getElementById("main");
main.style.top = window.getComputedStyle(main).top + 100 + "px";
// bad! JS
```

• a corrected version:

```javascript
main.style.top = parseInt(window.getComputedStyle(main).top) + 100 + "px"; // correct JS
```
Getting/setting CSS classes

```javascript
function highlightField() {
  // turn text yellow and make it bigger
  var text = document.getElementById("text");
  if (!text.className) {
    text.className = "highlight";
  } else if (!text.className.indexOf("invalid") < 0) {
    text.className += " highlight"; // awkward
  }
}
```

- JS DOM's className property corresponds to HTML class attribute
- somewhat clunky when dealing with multiple space-separated classes as one big string
Getting/setting CSS classes with classList

```javascript
function highlightField() {
    // turn text yellow and make it bigger
    var text = document.getElementById("text");
    if (!text.classList.contains("invalid")) {
        text.classList.add("highlight");
    }
}
```

- classList collection has methods add, remove, contains, toggle to manipulate CSS classes
- similar to existing className DOM property, but don't have to manually split by spaces
Removing a node from the page

```javascript
function slideClick() {
    var bullet = document.getElementById("removeme");
    bullet.parentNode.removeChild(bullet);
}
```

- odd idiom: `obj.parentNode.removeChild(obj);`
The keyword this

```javascript
// access field
this.fieldName = value;
// modify field
this.methodName(parameters);
// call method
```

- all JavaScript code actually runs inside of an object
- by default, code runs in the global window object (so this === window)
  - all global variables and functions you declare become part of window
- the this keyword refers to the current object

• all JavaScript code actually runs inside of an object
• by default, code runs in the global window object (so this === window)
  • all global variables and functions you declare become part of window
• the this keyword refers to the current object
Event handler binding

```javascript
window.onload = function() {
    document.getElementById("textbox").onmouseout = booyah;
    document.getElementById("submit").onclick = booyah;
};

function booyah() {
    // booyah knows what object it was called on
    this.value = "booyah";
}
```

- event handlers attached unobtrusively are bound to the element
- inside the handler, that element becomes this
Types of DOM nodes

• **element nodes** (HTML tag)
  • can have children and/or attributes

• **text nodes** (text in a block element)

• **attribute nodes** (attribute/value pair)
  • text/attributes are children in an element node
  • cannot have children or attributes
  • not usually shown when drawing the DOM tree

This is a paragraph of text with a link in it.
Traversing the DOM tree manually

every node's DOM object has the following properties:

<table>
<thead>
<tr>
<th>name(s)</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>firstChild, lastChild</td>
<td>start/end of this node's list of children</td>
</tr>
<tr>
<td>childNodes</td>
<td>array of all this node's children</td>
</tr>
<tr>
<td>nextSibling, previousSibling</td>
<td>neighboring nodes with the same parent</td>
</tr>
<tr>
<td>parentNode</td>
<td>the element that contains this node</td>
</tr>
</tbody>
</table>

- [complete list of DOM node properties](#)
- [browser incompatibility information](#) (IE6 sucks)
DOM tree traversal example

This is a paragraph of text with a link.

HTML
Element vs. text nodes

```html
<div>
  <p>
    This is a paragraph of text with a 
    <a href="page.html">link</a>.
  </p>
</div>
```

- **Q:** How many children does the div above have?
  - **A:** 3
    - an element node representing the `<p>`
    - two *text nodes* representing "\n\t" (before/after the paragraph)
- **Q:** How many children does the paragraph have? The `a` tag?