Element Attributes and Style

Lesson Objectives

When you complete this course, you will be able to:

- build the photo viewer application.
- use setAttribute to add a class to an element.
- use getAttribute to get the value of an element attribute.
- use document.querySelectorAll() and setAttribute() to set the class of multiple elements.
- set style directly using JavaScript.

One way you can use JavaScript to make your web applications more fun and interesting is to manipulate how your elements look on the page. In the next two lessons, we'll build a photo viewer web application. In this lesson, you'll learn how to use JavaScript to add and remove CSS style on your elements so you can show and hide menus. In the next lesson, you'll learn how to manipulate the DOM more so you can add and remove <img> elements on your page. Sound like fun? Let's do it!

First, check out the app you're going to build: The Photo Viewer App.

Make sure you're using the most recent version of whatever browser you're using. That means you should be using at least Internet Explorer 9, Safari 5, Chrome 16, Firefox 6, or Opera 11 (and by the time you read this, many of these browsers will have newer versions!).

Building the Photo Viewer Application

Start by creating the HTML for the Photo Viewer App. We'll put the JavaScript and CSS in separate files for this application, so make sure you notice the links to the CSS and JavaScript in the head of the document. You'll be creating those two files next. Make sure you save all three files in the *same folder*.

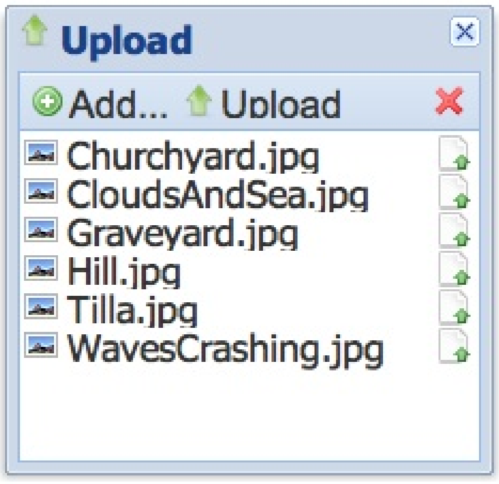
CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title> My Photos </title>
  <meta charset="utf-8">
  <link rel="stylesheet" href="photos.css">
  <script src="photos.js"></script>
</head>
<body>
  <nav>
    <ul>
      <li><span id="pets">Pets</span>
        <ul id="petsList">
          <li><a href="images/Tilla.jpg">Tilla</a></li>
          <li><a href="images/Pickles.jpg">Pickles</a></li>
          <li><a href="images/Jack.jpg">Jack</a></li>
        </ul>
      </li>
      <li><span id="landscape">Landscape</span>
         <ul id="landscapeList">
           <li><a href="images/Graveyard.jpg">Graveyard</a></li>
           <li><a href="images/Hill.jpg">Hill</a></li>
           <li><a href="images/Churchyard.jpg">Churchyard</a></li>
         </ul>
      </li>
      <li><span id="ocean">Ocean</span>
         <ul id="oceanList">
           <li><a href="images/WavesCrashing.jpg">Waves crashing</a></li>
           <li><a href="images/BreakingWave.jpg">Breaking wave</a></li>
           <li><a href="images/CloudsAndSea.jpg">Clouds and sea</a></li>
         </ul>
      </li>
    </ul>
  </nav>
  <div id="image">
  </div>
</body>
</html>
```

We're keeping this page very simple: we have a navigation section at the top with three nested menus, and a <div> where we'll place the image (next lesson!).

As you can see from the HTML, you're going to need images for this application. You can either use the photos from the example (to use these, right-click on the images and download them to your local machine) or use your own.

To add photos to your directory at OST, use the File Browser. Create a folder named **images** in the **/javascript1** folder. Select the **images** folder, and then click **File**. Click **Upload File**, and upload the photos from your local machine to the **images** folder.



Save it in your **/javascript1** folder as **photos.html** and click Preview to test the HTML. You haven't created the CSS and JavaScript yet, but you can test the links and make sure all your images are in the right places.

Add the CSS

Now it's time to add the CSS. Create a new file and add the CSS below.

Type the following into your HTML editor

```
body {
    font-family: helvetica, verdana, sans-serif;
}
ul {
    margin: 0px;
    padding: 0px;
    list-style: none;
}
ul li {
    display: inline-block;
    position: relative;
}
ul li span {
    padding: 2px 12px 2px 7px;
    background-color: #bebebe;
    display: block;
}
ul li ul {
    position: relative;
    margin: 0px;
    padding: 0px;
    left: 0px;
    top: 5px;
    width: 100%;
    visibility: hidden;
}
ul li ul li {
    padding: 2px 7px 2px 7px;
    display: block;
}
ul li ul li:hover {
    background-color: #bebebe;
}
ul li ul li a {
    display: block;
    color: black;
    text-decoration: none;
}
span:hover {
    cursor: hand;
    cursor: pointer;
    user-select: none;
}
.show {
    visibility: visible;
}
```

You don't need to understand all this CSS yet; we'll discuss the most important parts here.

OBSERVE:

```
ul li ul {
    position: relative;
    margin: 0px;
    padding: 0px;
    left: 0px;
    top: 5px;
    width: 100%;
    visibility: hidden;
}
```

ul li ul selects the nested lists; that is, the selector selects only those <ul> elements that are nested within <li> elements that are themselves nested within <ul> elements (you might want to refer back to the HTML). These nested lists are the drop-down parts of the menu, that appear below each of the main headings in the menu items: Pets, Landscape, Ocean. We want the initial state of the drop-down menus to be invisible, so we are using the **visibility** CSS property to hide them until you click on the main menu item. (Try the example again to see how this works.)

Save it in your **/javascript1** folder as **photos.css** so your link in the HTML will work correctly. Switch back to **photos.html** and click Preview. The CSS now takes effect, so you see only the main menu items, and the drop-down menus are invisible. In addition, you'll see that the menu items are in a horizontal layout, so the drop-down menus can "drop down" below each of the main items. However, when you click on the menu items, nothing happens; that's what we need the JavaScript to do!

Before we leave the CSS, also take note of the **.show** selector:

OBSERVE:

```
.show {
    visibility: visible;
}
```

You know, because of the "." in front of "show", that this is a class selector. All this rule does is set the visibility property back to visible. Now we'll write some JavaScript to add this class to the appropriate drop-down menu when you click on the corresponding main menu item.

Using setAttribute() to Add a Class to an Element

Next, we'll use the setAttribute() method to update the class of an element using JavaScript:

CODE TO TYPE:

```
window.onload = init;

function init() {
    var petsSpan = document.getElementById("pets");
    var landscapeSpan = document.getElementById("landscape");
    var oceanSpan = document.getElementById("ocean");

    petsSpan.onclick = selectPets;
    landscapeSpan.onclick = selectLandscape;
    oceanSpan.onclick = selectOcean;
}

function selectPets() {
    var ul = document.getElementById("petsList");
    ul.setAttribute("class", "show");
}

function selectLandscape() {
    var ul = document.getElementById("landscapeList");
    ul.setAttribute("class", "show");
}

function selectOcean() {
    var ul = document.getElementById("oceanList");
    ul.setAttribute("class", "show");
}
```

Save it in your **/javascript1** folder as **photos.js**, switch back to **photos.html**, and click Preview. Click on the main menu items. The drop-down menus now appear! And, because each item in the drop-down menus is a link, when you click on the image items, you'll see a new page displaying the image. We're going to change this behavior in the next lesson, so for now just use the back button to come back to the main page.

Let's walk through how the JavaScript works.

OBSERVE:

```
function selectPets() {
    var ul = document.getElementById("petsList");
    ul.setAttribute("class", "show");
}
```

If you look back at the HTML, you'll notice that each of the main menu items has an id in a <span> element, and each of the drop-down menus also has an id in the <ul> element for that menu. So, we first add a click handler function to each of the <span> elements; and when, say, the "Pets" menu item is clicked (that is, you click on the <span>), the appropriate click handler is called (in this case, selectPets()), which gets the drop-down menu for Pets by its id, "petsList," and then uses the **setAttribute()** method to set the **class** attribute of that <**ul**> element to the **"show"** class. By setting the **class** to **"show"**, we override the **visibility** property that has the <ul> set to "hidden", with a new visibility property that makes it visible (remember that CSS rules that come *later* in the file override properties in *earlier* rules).

Now let's take a closer look at the **setAttribute()** method. This method is another element object method (like appendChild() that we used in the previous lesson), so you call it on the element whose attribute you want to set. The first argument is the name of the attribute, in this case, **"class"**, and the second argument is the value for the attribute, in this case, **"show"**. By setting the class attribute in this way, it's as if you'd typed:

<ul id="petsList" class="show">

Note that when you set the **class** attribute like this, you'll overwrite any previous value of the class attribute that element might have had. In our case, that doesn't matter, but there may be times when you want to *add*, rather than *replace*, a class, and for that, you'll need to first get the value of the attribute and then add on the new class name! (Sounds like a good exercise for the reader...).

Using getAttribute() to Get the Value of an Element Attribute

You might have already noticed that once you've clicked on a main menu item, the drop-down menu doesn't go away! What we'd really like is to be able to click on the main menu item a second time to make the drop-down menu disappear (this is commonly how menu items work). To implement this functionality, we need to be able to determine if a drop-down menu is visible or not when you click on the main menu item. If it is visible already, then we need to make it invisible. However, if it's invisible, then we do what we're already doing: make it visible. We can do this by checking to see if the "show" class is set on the <ul> for the drop-down menu; if it's there, we know the drop-down menu is visible, and we can make it invisible again by *removing* the "show" class from the element.

To do all that, we'll need to make use of the **getAttribute()** method. Update your "photos.js" file:

CODE TO TYPE:

```
window.onload = init;

function init() {
    var petsSpan = document.getElementById("pets");
    var landscapeSpan = document.getElementById("landscape");
    var oceanSpan = document.getElementById("ocean");

    petsSpan.onclick = selectPets;
    landscapeSpan.onclick = selectLandscape;
    oceanSpan.onclick = selectOcean;
}

function selectPets() {
    var ul = document.getElementById("petsList");
    ~~ul.setAttribute("class", "show");~~
    showHide(ul);
}

function selectLandscape() {
    var ul = document.getElementById("landscapeList");
    ~~ul.setAttribute("class", "show");~~
    showHide(ul);
}

function selectOcean() {
    var ul = document.getElementById("oceanList");
    ~~ul.setAttribute("class", "show");~~
    showHide(ul);
}

function showHide(el) {
    var ulClass = el.getAttribute("class");
    if (ulClass == "show") {
        // item is selected, so deselect it
        el.setAttribute("class", "");
    }
    else {
        // item is not selected, so select it
        el.setAttribute("class", "show");
    }
}
```

Save it, switch back to **photos.html**, and click Preview. Clicking once on a main menu item to display the drop-down menu, and then click again on the same main menu item—the drop-down menu should disappear. Try it with each of the main menu items.

To implement this, we consolidated all the new code into a new function, **showHide()**, since it's the same code in each case, and we pass in the correct <ul> element from each of the click handlers. So, if you click on "Pets," we'll pass in the "petsList" <ul> element.

OBSERVE:

```
function showHide(el) {
    var ulClass = el.getAttribute("class");
    if (ulClass == "show") {
        // item is selected, so deselect it
        el.setAttribute("class", "");
    }
    else {
        // item is not selected, so select it
        el.setAttribute("class", "show");
    }
}
```

First, we **get the current value of the class attribute using the getAttribute() method**. Again, this is an element object method; so we call it on the element whose attribute we want to get—if you click on **Pets**, that element is "petsList" <ul>. The first time you click on **Pets**, the "petsList" <ul> has no class at all, so the **ulClass** variable will be "". We **check to see if the ulClass variable contains the "show" string**, which it doesn't this time, so we do what we were doing before: we **set the class attribute of the "petsList" <ul> to "show."** However, if you click Pets again when the class attribute is already set to "show", we want to *remove* the class, so the drop-down menu will disappear. To do this, we can use **setAttribute()** again, and **set the value of the class attribute to ""**. As soon as this code runs, the "visibility" is again set to "hidden" and the drop-down menu disappears!

Use document.querySelectorAll() and setAttribute() to Set the Class of Multiple Elements

There's one more thing to do to this application before we end the lesson. Did you notice that if you click on one main menu item, like **Pets**, and then click on another, like **Landscape**, the **Pets** drop-down menu is still visible? The behavior we'd like (and the behavior we are more likely to expect) is that if you click on **Pets** and then **Landscape**, the **Pets** drop-down menu will disappear. Try the completed example again if you want to compare this behavior to the current behavior of your application.

To make this work, we need to add one more thing to the code. When you click on a main menu item like **Pets**, before we determine if we should show or hide the "Pets" menu, we need to make sure the other drop-down menus are hidden. To do this, we'll use a method to select elements from the DOM that you haven't seen before: **document.querySelectorAll()**. This method, which is new in HTML5, makes it super-easy to select elements from the DOM using CSS-like selectors.

Note The **document.querySelectorAll()** method is new in HTML5 so it will only work in modern browsers. Make sure you're using the most recent version of your browser.

CODE TO TYPE:

```
window.onload = init;

function init() {
    var petsSpan = document.getElementById("pets");
    var landscapeSpan = document.getElementById("landscape");
    var oceanSpan = document.getElementById("ocean");

    petsSpan.onclick = selectPets;
    landscapeSpan.onclick = selectLandscape;
    oceanSpan.onclick = selectOcean;
}

function selectPets() {
    var ul = document.getElementById("petsList");
    showHide(ul);
}

function selectLandscape() {
    var ul = document.getElementById("landscapeList");
    showHide(ul);
}

function selectOcean() {
    var ul = document.getElementById("oceanList");
    showHide(ul);
}

function showHide(el) {
    // hide everything except what we clicked on
    var selectedItems = document.querySelectorAll(".show");
    for (var i = 0; i < selectedItems.length; i++) {
        if (selectedItems[i] != el) {
            selectedItems[i].setAttribute("class", "");
        }
    }

    var ulClass = el.getAttribute("class");
    if (ulClass == "show") {
        // item is selected, so deselect it
        el.setAttribute("class", "");
    }
    else {
        // item is not selected, so select it
        el.setAttribute("class", "show");
    }
}
```

Save it, switch back to **photos.html**, and click Preview. Click **Pets**, then click **Landscape**. The Pets drop-down menu should disappear. Now click **Ocean**, and the Landscape drop-down should disappear.

Let's step through an example to see how this works. Suppose you clicked the **Pets** main menu item, so the "petsList" drop-down menu is visible. That means that the "petsList" <ul> element has a class attribute set to "show." Now, you click **Landscape**, the selectLandscape() method is called, and that method calls showHide(), and passes in the "landscapeList" <ul> element.

OBSERVE:

```
function showHide(el) {
    // hide everything except what we clicked on
    var selectedItems = document.querySelectorAll(".show");
    for (var i = 0; i < selectedItems.length; i++) {
        if (selectedItems[i] != el) {
            selectedItems[i].setAttribute("class", "");
        }
    }

    var ulClass = el.getAttribute("class");
    if (ulClass == "show") {
        // item is selected, so deselect it
        el.setAttribute("class", "");
    }
    else {
        // item is not selected, so select it
        el.setAttribute("class", "show");
    }
}
```

Before adding the "show" class to the "landscapeList" <ul> element, we want to remove the "show" class from the "petsList" <ul> element. To do this, we have to find out which element has the "show" class so we can remove it. To get all of the elements that currently have the class "show," we use the **querySelectorAll()** method.

This method takes a string, which is a selector, similar to what you use in CSS to select elements for styling. We use the string **".show"** to select all the elements with the class "show." No matter what state the application is in, we should only get zero or one elements as a result, but no matter how many we get back, the result is always a collection of elements (which can be empty if *no* elements have that class). In this case, however, we know that the "petsList" <ul> element does have this class, so we'll get that element back as the result.

We **loop through all the items in the collection (which is just one)**, and we **compare each item to the one we just clicked on**. We only want to remove the "show" class from the elements that currently have it, but *not* if the element is the one we've just clicked on. Why? Because if you click on the *same* menu item (that is, instead of clicking **Landscape**, you click on **Pets** to close the drop-down menu), you'll be removing the "show" class and then the remaining code in the function will add the "show" class again, and you'll be in a state where you can never get the Pets menu to go away!

If we find a drop-down menu that is *different* from the one we've just clicked on that has the class "show," we remove the class just like we did previously, by **using setAttribute() to set the class attribute to ""**. So, we remove the "show" class from the "petsList" drop-down menu, which hides it. Now we're in a state where no drop-downs are displayed.

The rest of the code is the same. We get the class attribute for the drop-down menu corresponding to the menu item we clicked; in our example, that's the "landscapeList" <ul> element. This element does *not* have the "show" class (because we hadn't clicked on it previously), so we set the class attribute to "show" and it appears.

We're using the "show" class as a way to detect whether a menu item is visible or not. This is a common thing to do in web applications, and so you'll probably find yourself using getAttribute() and setAttribute() a lot to do similar things.

Try stepping through another couple of use cases to make sure you understand how the code works when the menus are in various different states.

Keep this code because you'll be using it in the next lesson!

Setting Style Directly Using JavaScript

You know how to change the style of an element using JavaScript to add or remove a class attribute from that element. Another way to manipulate the style of an element is to use the element object's **style** property. The style property is actually an object that contains some (but not all!) of the properties you can set in CSS as JavaScript properties. Because not all CSS properties are available in the style object, it's not quite as flexible as being able to set the class attribute to an arbitrary class with any CSS properties you want, but it can still come in handy!

Let's take a look at how you can set **style** directly using JavaScript, rather than by adding or removing **classes**. Neither method is better than the other, although using **classes** is often more concise, which is always a good thing.

CODE TO TYPE:

```
<!doctype html>
<html>
  <head>
  <title>Setting Style with JavaScript</title>
  <meta charset="utf-8">
  <style>
    div#container {
        display: table;
        border-spacing: 10px;
    }
    div.box {
        color: white;
        width: 200px;
        height: 200px;
        padding: 10px;
    }
    div#div1 {
        display: table-cell;
        background-color: blue;
    }
    div#div2 {
        display: table-cell;
        background-color: darkgreen;
    }
    div#div3 {
        display: table-cell;
        background-color: purple;
    }
  </style>
  <script>
    window.onload = init;

    function init() {
        var div1 = document.getElementById("div1");
        div1.onclick = changeVisibility;

        var div2 = document.getElementById("div2");
        div2.onclick = changeColor;

        var div3 = document.getElementById("div3");
        div3.onclick = changeBorder;
    }

    function changeVisibility() {
        var div1 = document.getElementById("div1");
        div1.style.visibility = "hidden";
    }
    function changeColor() {
        var div2 = document.getElementById("div2");
        div2.style.backgroundColor = "red";
        div2.style.color = "black";
    }
    function changeBorder() {
        var div3 = document.getElementById("div3");
        div3.style.borderWidth = "5px";
        div3.style.borderColor = "black";
        div3.style.borderStyle = "solid";
    }
  </script>
</head>
<body>
<div id="container">

<div class="box" id="div1">
Click me to make me invisible!
</div>

<div class="box" id="div2">
Click me to change my color!
</div>

<div class="box" id="div3">
Click me to change my border!
</div>

</div>
</body>
</html>
```

Save the file in the **/javascript1** folder as **style.html** and click Preview. Try clicking on the different boxes. The first box disappears, the second box changes color, and the third box changes its border.

Let's step through how this works.

OBSERVE:

```
function init() {
    var div1 = document.getElementById("div1");
    div1.onclick = changeVisibility;

    var div2 = document.getElementById("div2");
    div2.onclick = changeColor;

    var div3 = document.getElementById("div3");
    div3.onclick = changeBorder;
}
```

First, we **set click handlers for each of the <div> elements**. Each <div> gets a different click handler, because each function will do something different.

Let's check out the **changeVisibility()** function first. When you click on "div1," the <div> disappears. We do this by setting the **visibility** CSS property using the **style** property of the **<div>** object:

OBSERVE:

```
function changeVisibility() {
    var div1 = document.getElementById("div1");
    div1.style.visibility = "hidden";
}
```

Notice that we use the string **"hidden"** to change the value of the **visibility** property. This is the same exact value you'd use for the visibility property in CSS if you were doing this using CSS.

When you click on the second <div> the color changes (both the background color and the text color).

OBSERVE:

```
function changeColor() {
    var div2 = document.getElementById("div2");
    div2.style.backgroundColor = "red";
    div2.style.color = "black";
}
```

We get the "div2" **<div>** object, and then set the **backgroundColor** and **color** properties again using the **style**. Again, notice that the **values** we set these properties to are exactly the values you'd use if you were setting them using CSS.

However, notice that the **backgroundColor** property name is a little different from what you'd use in CSS. In CSS, you set the background color of an element using the property name "background-color". But in JavaScript, background-color is not a valid variable name, right?! So, the name of the property in JavaScript is **style**.**backgroundColor**. This is common; for instance, font-size becomes **style.fontSize**, border-width becomes **style.borderWidth** and so on.

OBSERVE:

```
function changeBorder() {
    var div3 = document.getElementById("div3");
    div3.style.borderWidth = "5px";
    div3.style.borderColor = "black";
    div3.style.borderStyle = "solid";
}
```

The changeBorder() function is similar to the others. Notice that when you set the **borderWidth** property to be 5 pixels wide, you must include "px" in the **property value**, just like you would in CSS.

Experiment! Try setting the style values of various CSS properties on elements using JavaScript. It's kind of fun!