

Advanced JavaScript Programming

Dynamic HTML

Lesson 1, Activity 2: **Introduction**

Dynamic HTML is not a technology in and of itself, but rather is a combination of three technologies: HTML, Cascading Style Sheets (CSS), and JavaScript. It usually involves using JavaScript to change a CSS property of an HTML element. In modern browsers, most CSS properties can be modified dynamically. This can be done by changing an individual style of an element (using the `style` property of the element) or by changing the class name assigned to the element (using the `className` property).

Lesson 1, Activity 3: Accessing and Modifying Styles

The `style` object is a collection of an element's styles that are either defined within that HTML element's `style` attribute or directly in JavaScript. Styles defined in the `<style>` tag or in an external style sheet are not part of the `style` object.

The W3C specifies a method for getting at the current (or actual) style of an object: the `window` object's `getComputedStyle()` method, which is supported by most modern browsers, but not by Internet Explorer 6 and earlier. Internet Explorer provides a non-standard property for getting at the current style of an element: the `currentStyle` property.

Standard Syntax

```
window.getComputedStyle(Element, Pseudo-Element)
```

Note that the reference to `window` can be excluded as `window` is the implicit object. For example:

```
var div = document.getElementById("divTitle");
var curStyle = getComputedStyle(div, null);
alert(curStyle.fontWeight);
```

Internet Explorer 6 Syntax

```
Element.currentStyle.Property
```

For example:

```
var div = document.getElementById("divTitle");
var curStyle = div.currentStyle;
alert(curStyle.fontWeight);
```

Cross-browser Function

The following cross-browser function for getting the current value of a specific style property of an element is included in our [lib.js](#) file:

```
function getCurrentStyle(elem,property) {
  if (window.getComputedStyle) {
    var computedStyle = getComputedStyle(elem,null);
    return computedStyle[property];
  } else if (elem.currentStyle) {
    return elem.currentStyle[property];
  }
}
```

JavaScript style Properties

Each CSS property has a corresponding property of the JavaScript `style` object:

- If the CSS property is a simple word (e.g, `color`) then the JavaScript property is the same (e.g, `style.color`).
- If the CSS property has a dash in it (e.g, `background-color`) then the JavaScript property uses lower camel case (e.g, `style.backgroundColor`).

Lesson 1, Activity 4: Hiding and Showing Elements

Elements can be hidden and shown by changing their `visibility` or `display` values. The `visibility` style can be set to `visible` or `hidden` and the `display` property can be set to `block`, `table-row`, `none`, and several other values. The two work slightly differently as the following example illustrates:

Code Sample:

[DynamicHtml/Demos/Visibility.html](#)

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="UTF-8">
<title>Showing and Hiding Elements with JavaScript</title>
<link href="visibility.css" rel="stylesheet" type="text/css">
<script type="text/javascript" src="../../lib.js"></script>
<script type="text/javascript">
function changeVisibility(id){
    var elem = document.getElementById(id);
    var visibility;
    if (elem.style.visibility=="hidden") {
        elem.style.visibility = "visible";
    } else {
        elem.style.visibility = "hidden";
    }
    visibility = getComputedStyle(elem,"visibility");
    msg("The <em>visibility</em> of row <em>" + id + "</em> is <em>" + visibility + "</em>.");
}

function changeDisplay(id){
    var elem = document.getElementById(id);
    var display;
    if (elem.style.display == "none") {
        elem.style.display = "";
    } else {
        elem.style.display = "none";
    }
    display = getComputedStyle(elem,"display");
    msg("The <em>display</em> of row <em>" + id + "</em> is <em>" + display + "</em>.");
}

function msg(text) {
    document.getElementById("msg").innerHTML = text;
}
</script>
</head>
<body>
<h1>Hiding and Showing Elements</h1>
<table>
  <tr id="tr1"><td>Row 1</td></tr>
  <tr id="tr2"><td>Row 2</td></tr>
  <tr id="tr3"><td>Row 3</td></tr>
  <tr id="tr4"><td>Row 4</td></tr>
</table>
<div id="msg">Style messages</div>
<h2>visibility</h2>
```

```

<button onclick="changeVisibility('tr1')">Row 1</button>
<button onclick="changeVisibility('tr2')">Row 2</button>
<button onclick="changeVisibility('tr3')">Row 3</button>
<button onclick="changeVisibility('tr4')">Row 4</button>

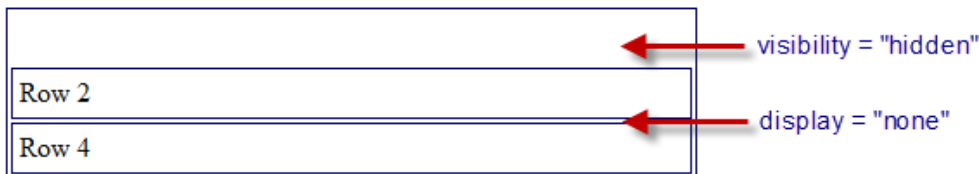
<h2>display</h2>
<button onclick="changeDisplay('tr1')">Row 1</button>
<button onclick="changeDisplay('tr2')">Row 2</button>
<button onclick="changeDisplay('tr3')">Row 3</button>
<button onclick="changeDisplay('tr4')">Row 4</button>
</body>
</html>

```

This page has two functions: `changeVisibility()` and `changeDisplay()`. The `changeVisibility()` function checks the value of the `visibility` style of the passed element and changes it to its opposite. The `changeDisplay()` function does the same with the `display` style. The functions are called with buttons on the page and are passed in ids of table rows from the table on the page.

The main difference between setting `visibility` to `hidden` and setting `display` to `none` is that setting `visibility` to `hidden` does not modify the layout of the page; it simply hides the element. Setting `display` to `none`, on the other hand, collapses the element, so that the surrounding relatively positioned elements re-position themselves.

Check out the screenshot below:



Row 1 has `visibility` set to `hidden`, so you can see the space for the row, but you cannot see the row itself. Row 3 has `display` set to `none` so it's as if the row was not even there.

The following line of code may have grabbed your attention:

```
elem.style.display = "";
```

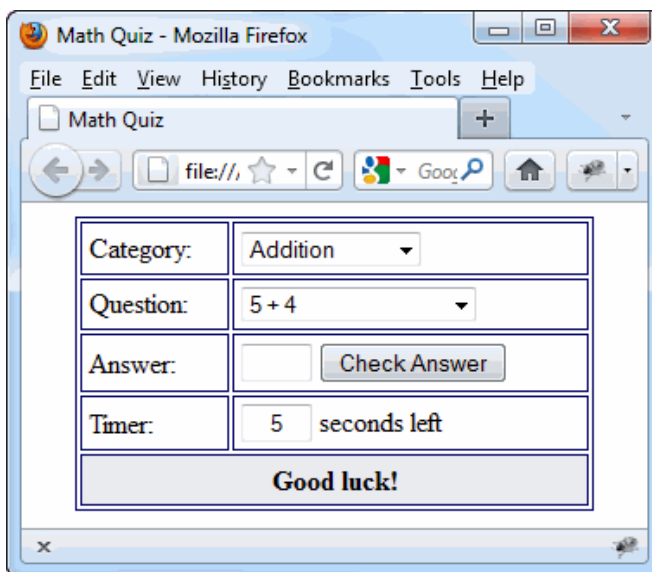
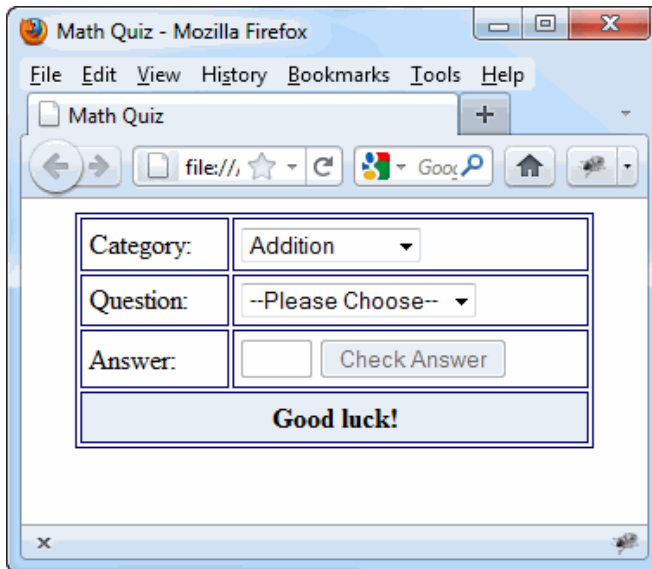
According to the CSS specification, the proper way to display a table row is by setting the `display` property to `table-row`; however Internet Explorer 6 and earlier do not support this value. To get the script to behave correctly in IE6, the `display` must be set to `block`; however, that messes up standard browsers, which display blocks as blocks, not as table rows. One solution is to set the value to an empty string ("") as in the line of code above. That tells the browser to display the table row as it normally would without any display styling at all.

Lesson 1, Activity 6: Showing and Hiding Elements

Duration: 20 to 30 minutes.

In this exercise, you will modify a Math Quiz to only show the countdown timer when it is running.

1. Open <DynamicHtml/Exercises/MathQuiz.html> for editing.
2. Modify the code so that the table row with the timer in it (the last row) only shows up when the timer is running.



Hint: You will make your changes in `resetTimer()` and `questionChanged()`.

3. Test your solution in a browser.

Challenge

Only show the Answer row when a question is selected.

Solution:

<DynamicHtml/Solutions/MathQuiz.html>

---- C O D E O M I T T E D ----

```
function resetTimer(seconds) {
    var timerRow = document.getElementById("timerRow");
    timerRow.style.display="none";
    document.getElementById("timeLeft").value = seconds;
    clearInterval(timer);
}
```

---- C O D E O M I T T E D ----

```
function questionChanged() {
    document.Quiz.answer.value="";
    var timerRow;
    if (document.Quiz.question.selectedIndex === 0) {
        document.Quiz.btnCheck.disabled = true;
        document.Quiz.answer.disabled = true;
        resetTimer(timePerQuestion);
    } else {
        document.Quiz.btnCheck.disabled = false;
        document.Quiz.answer.disabled = false;
        document.Quiz.answer.focus();
        timer = setInterval(decrementTimer,1000);
        timerRow = document.getElementById("timerRow");
        timerRow.style.display="";
    }
}
```

---- C O D E O M I T T E D ----

```
<tr id="timerRow">
    <td>Timer:</td>
    <td><input type="text" name="timeLeft" id="timeLeft" size="2"> seconds left</td>
</tr>
<tr>
    <td id="msg" colspan="2">Good luck!</td></tr>
</tr>
</table>
</form>
</body>
</html>
```

Challenge Solution:

DynamicHtml/Solutions/MathQuiz-challenge.html

---- C O D E O M I T T E D ----

```
function resetTimer(seconds) {
    var timerRow = document.getElementById("timerRow");
    var answerRow = document.getElementById("answerRow");
    timerRow.style.display="none";
    answerRow.style.display="none";
    document.getElementById("timeLeft").value = seconds;
    clearInterval(timer);
}
```


---- C O D E O M I T T E D ----

```
function questionChanged() {
    document.Quiz.answer.value="";
    var timerRow,answerRow;
    if (document.Quiz.question.selectedIndex === 0) {
        document.Quiz.btnCheck.disabled = true;
        document.Quiz.answer.disabled = true;
        resetTimer(timePerQuestion);
    } else {
        document.Quiz.btnCheck.disabled = false;
        document.Quiz.answer.disabled = false;
        timer = setInterval(decrementTimer,1000);
        timerRow = document.getElementById("timerRow");
        timerRow.style.display="";
        answerRow = document.getElementById("answerRow");
        answerRow.style.display="";
        document.Quiz.answer.focus();
    }
}
```

---- C O D E O M I T T E D ----

```
<tr id="answerRow">
```

---- C O D E O M I T T E D ----

Lesson 1, Activity 7: Manipulating Tables

HTML tables can be created and manipulated dynamically with JavaScript. Each `table` element contains a `rows` array and methods for inserting and deleting rows: `insertRow()` and `deleteRow()`. Each `tr` element contains a `cells` array and methods for inserting and deleting cells: `insertCell()` and `deleteCell()`. The following example shows how these objects can be used to dynamically create HTML tables:

Code Sample:

[DynamicHtml/Demos/Table.html](#)

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="UTF-8">
<title>Manipulating Tables</title>
<link href="table.css" rel="stylesheet" type="text/css">
<script type="text/javascript" src="../../lib.js"></script>
<script type="text/javascript">
function addRow(tableId, cells){
    var tableElem = document.getElementById(tableId);
    var newRow = tableElem.insertRow(tableElem.rows.length);
    var newCell;
    for (var i = 0; i < cells.length; i++) {
        newCell = newRow.insertCell(newRow.cells.length);
        newCell.innerHTML = cells[i];
    }
    return newRow;
}

function deleteRow(tableId, rowNum){
    var tableElem = document.getElementById(tableId);
    if (rowNum > 0 && rowNum < tableElem.rows.length) {
        tableElem.deleteRow(rowNum);
    } else {
        alert("Failed");
    }
}

observeEvent(window,"load",function() {
    var btnAdd = document.getElementById("btnAdd");
    var btnDelete = document.getElementById("btnDelete");
    observeEvent(btnAdd,"click",function() {
        var cells=[btnAdd.form.FirstName.value,btnAdd.form.LastName.value];
        addRow('tblPeople', cells);
    });
    observeEvent(btnDelete,"click",function() {
        deleteRow('tblPeople', btnDelete.form.RowNum.value)
    });
});
</script>
</head>
<body>
<table id="tblPeople">
<tr>
    <th>First Name</th>
    <th>Last Name</th>
```

```

</tr>
</table>
<hr>
<form name="formName">
  <label for="FirstName">First Name:</label>
  <input type="text" name="FirstName"><br>
  <label for="LastName">Last Name:</label>
  <input type="text" name="LastName"><br>
  <input type="button" id="btnAdd" value="Add Name">
  <hr>
  <label for="RowNum">Remove Row:</label>
  <input type="text" size="1" name="RowNum">
  <input type="button" id="btnDelete" value="Delete Row">
</form>
</body>
</html>

```

The body of the page contains a table with an id of `formName`. The table contains a single row of headers.

```

<table id="tblPeople">
  <tr>
    <th>First Name</th>
    <th>Last Name</th>
  </tr>
</table>

```

Below the table is a form that allows the user to enter a first and last name. When the "Add Name" button is clicked, the `addRow()` function is called and passed in the id of the table (`tblPeople`) and a new array containing the user-entered values:

```

observeEvent(btnAdd,"click",function() {
  var cells = [btnAdd.form.FirstName.value, btnAdd.form.LastName.value];
  addRow('tblPeople', cells);
});

```

The `addRow()` function uses the `insertRow()` method of the `table` to add a new row at the end of the table and then loops through the passed-in array, creating and populating one cell for each item. The function also returns the new row. Although the returned value isn't used in this example, it can be useful if you then want to manipulate the new row further. This function is included in our [lib.js](#) file.

```

function addRow(tableId, cells){
  var tableElem = document.getElementById(tableId);
  var newRow = tableElem.insertRow(tableElem.rows.length);
  var newCell;
  for (var i = 0; i < cells.length; i++) {
    newCell = newRow.insertCell(newRow.cells.length);
    newCell.innerHTML = cells[i];
  }
  return newRow;
}

```

The form also contains a "Delete Row" button that, when clicked, passes the id of the table (`tblPeople`) and the number entered by the user in the `RowNum` text field.

```
observeEvent(btnDelete, "click", function() {  
  deleteRow('tblPeople', btnDelete.form.RowNum.value)  
});
```

The `deleteRow()` function checks to see if the row specified exists and is not the first row (row 0, which is the header row). If both conditions are true, it deletes the row. Otherwise, it alerts "Failed".

```
function deleteRow(tableId, rowNumber){  
  var tableElem = document.getElementById(tableId);  
  if (rowNumber > 0 && rowNumber < tableElem.rows.length) {  
    tableElem.deleteRow(rowNumber);  
  } else {  
    alert("Failed");  
  }  
}
```

Lesson 1, Activity 8: Tracking Results in the Math Quiz

Duration: 15 to 25 minutes.

In this exercise, you will dynamically create a table that shows the user how she is doing on the Math Quiz. The screenshot below shows how the result will look:

Category:	Subtraction ▼
Question:	--Please Choose-- ▼
Sorry. The correct answer is 8.	

Question	Your answer	Correct answer
5 + 4	9	9
9 + 3	12	12
7 + 12	19	19
13 + 24	36	37
5 - 1	4	4
12 - 4	7	8

1. Open [DynamicHtml/Exercises/MathQuizTable.html](#) for editing.
2. Change the `msg()` function so that it takes a second argument: `color`, and uses it to change the text of the "msg" div to the passed-in color.
3. Change all calls to `msg()` to pass in a `color` as well as the `text`: green if the answer is correct and red if it is not.
4. Option objects have a `text` property that holds the displayed text of the option. Add a `getQuestion()` function that returns the text of the selected question.
5. Notice that there is a table at the bottom of the body of the page with an `id` of `tblResults`.
6. Remember that the `addRow()` function is in the [lib.js](#) file. Modify the `checkAnswer()` function so that it adds a new row to the results table showing the question, user's answer, and correct answer.
7. Test your solution in a browser.

Challenge

Modify the code so that the new row's background color is `#00ff00` if the answer is correct and `#ff9999` if it is not.

Hint: the `addRow()` function returns the newly added row.

Solution:

[DynamicHtml/Solutions/MathQuizTable.html](#)

---- C O D E O M I T T E D ----

```
function checkAnswer() {
    var userAnswer = document.getElementById("answer").value;
    var correctAnswer = getAnswer();
```

```

var question = getQuestion();
var arrCells = [question, userAnswer, correctAnswer];
addRow("tblResults", arrCells);

if (userAnswer === correctAnswer) {
    msg("Right! The answer is " + correctAnswer + ".", "green");
} else {
    msg("Sorry. The correct answer is " + correctAnswer + ".", "red");
}
removeOption();
questionChanged();
}

function getAnswer() {
    var i = document.Quiz.question.selectedIndex;
    var answer = document.Quiz.question[i].value;
    return answer;
}

function getQuestion(){
    var i = document.Quiz.question.selectedIndex;
    var question = document.Quiz.question[i].text;
    return question;
}

---- C O D E   O M I T T E D ----

function msg(text, color) {
    document.getElementById("msg").innerHTML=text;
    document.getElementById("msg").style.color = color;
}
</script>
</head>
<body>

---- C O D E   O M I T T E D ----

<table id="tblResults">
<tr>
    <th>Question</th>
    <th>Your answer</th>
    <th>Correct answer</th>
</tr>
</table>
</body>
</html>

```

Challenge Solution:

[DynamicHtml/Solutions/MathQuizTable-challenge.html](#)

```

---- C O D E   O M I T T E D ----

function checkAnswer() {
    var userAnswer = document.getElementById("answer").value;
    var correctAnswer = getAnswer();
    var question = getQuestion();
    var arrCells = [question, userAnswer, correctAnswer];
    var row = addRow("tblResults", arrCells);
}

```

```
if (userAnswer === correctAnswer) {  
    msg("Right! The answer is " + correctAnswer + ".", "green");  
    row.style.backgroundColor="#00ff00";  
} else {  
    msg("Sorry. The correct answer is " + correctAnswer + ".", "red");  
    row.style.backgroundColor="#ff9999";  
}  
removeOption();  
questionChanged();  
}  
---- C O D E    O M I T T E D ----
```

Lesson 1, Activity 10: Dynamically Changing Dimensions

The dimensions of an object can be changed by modifying the width and height properties of the element's style property. The following example demonstrates this:

Code Sample:

DynamicHtml/Demos/Dimensions.html

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="UTF-8">
<title>Dimensions</title>
<link href="dimensions.css" rel="stylesheet" type="text/css">
<script type="text/javascript" src="../../lib.js"></script>
<script type="text/javascript">
function grow(elem){
    var curWidth = parseInt( getComputedStyle(elem,"width") );
    var curHeight = parseInt( getComputedStyle(elem,"height") );
    elem.style.width = (curWidth * 1.5) + 'px';
    elem.style.height = (curHeight * 1.5) + 'px';
    showDimensions(elem);
}

function shrink(elem){
    var curWidth = parseInt( getComputedStyle(elem,"width") );
    var curHeight = parseInt( getComputedStyle(elem,"height") );
    elem.style.width = (curWidth / 1.5) + 'px';
    elem.style.height = (curHeight / 1.5) + 'px';
    showDimensions(elem);
}

function showDimensions(elem) {
    elem.innerHTML = "w: " + getComputedStyle(elem,"width") + "<br>h: " + getComputedStyle(elem,"height");
}

observeEvent(window,"load",function() {
    var block = document.getElementById("divBlock");
    observeEvent(block,"mouseover",function() {
        grow(block);
    })
    observeEvent(block,"mouseout",function() {
        shrink(block);
    })
    showDimensions(block);
});
</script>
</head>
<body>
<div id="divBlock"></div>
</body>
</html>
```

When the page loads, we begin observing mouseover and mouseout events on the block div. We call grow() on

mouse overs and `shrink()` on mouse outs.

The `grow()` function uses `parseInt()` to cut off the units (e.g, px) from the value of the width and height of the div and assigns the resulting integers to variables: `curWidth` and `curHeight`. It then modifies the width and height properties of the element by multiplying the current values by 1.5.

The `shrink()` function does the same thing, but it divides by 1.5 instead of multiplying.

Creating a Timed Slider

The example below shows how a timed slider can be created by dynamically changing an element's dimensions:

Code Sample:

[DynamicHtml/Demos/Slider.html](#)

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="UTF-8">
<title>Slider</title>
<link href="slider.css" rel="stylesheet" type="text/css">
<script type="text/javascript" src="../../lib.js"></script>
<script type="text/javascript">
    var timer, timesUp;
    function resetTimer() {
        var slider = document.getElementById("divSlider");
        timesUp = true;
        slider.style.width = "0px";
        clearInterval(timer);
    }

    function decrementTimer() {
        var slider = document.getElementById("divSlider");
        var curWidth = parseInt( getComputedStyle(slider,"width") );
        timesUp = false;
        if (curWidth < 200) {
            slider.style.width = curWidth + 2 + "px";
        } else {
            alert("Time's up!");
            resetTimer();
        }
    }

    observeEvent(window,"load",function() {
        var btnStart = document.getElementById("btnStart");
        resetTimer();
        observeEvent(btnStart,"click",function() {
            timer = setInterval(decrementTimer, 100);
        });
    });
</script>
</head>
<body>
<div id="divSliderBG">
    <div id="divSlider"></div>
</div>
```

```
<button id="btnStart">Start Timer</button>
</body>
</html>
```

Lesson 1, Activity 12: Positioning Elements Dynamically

The position of an object can be changed by modifying the `left` and `top` properties of the element's `style` property. The following example demonstrates this:

Code Sample:

DynamicHtml/Demos/Position.html

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="UTF-8">
<title>Position</title>
<link href="position.css" rel="stylesheet" type="text/css">
<script type="text/javascript" src="../../lib.js"></script>
<script type="text/javascript">
  function moveH(elem, distance){
    var curLeft = parseInt( getComputedStyle(elem,"left") );
    elem.style.left = (curLeft + distance) + "px";
  }

  function moveV(elem, distance){
    var curTop = parseInt( getComputedStyle(elem,"top") );
    elem.style.top = (curTop + distance) + "px";
  }

  observeEvent(window,"load",function() {
    var btnLeft = document.getElementById("btnLeft");
    var btnRight = document.getElementById("btnRight");
    var btnUp = document.getElementById("btnUp");
    var btnDown = document.getElementById("btnDown");
    var block = document.getElementById("divBlock");

    observeEvent(btnLeft,"click",function() {
      moveH(block,-10);
    });
    observeEvent(btnRight,"click",function() {
      moveH(block,10);
    });
    observeEvent(btnUp,"click",function() {
      moveV(block,-10);
    });
    observeEvent(btnDown,"click",function() {
      moveV(block,10);
    });
  });
</script>
</head>
<body>
<div id="field">
  <button id="btnLeft">Left</button>
  <button id="btnRight">Right</button>
  <button id="btnUp">Up</button>
  <button id="btnDown">Down</button>
  <div id="divBlock"></div>
</div>
</body>
```

```
</html>
```

When the page loads, we begin observing `click` events on the buttons to call `moveH()` and `moveV()`.

The `moveH()` function uses `parseInt()` to cut off the units (e.g. `px`) from the value of the `left` property and assigns the resulting integers to the `curLeft` variable. It then adds the passed-in distance to the `left` property.

The `moveV()` function does the same thing, but it modifies the `top` property rather than the `left` property.

Creating a Different Timed Slider

The example below shows how a different type of timed slider can be created by dynamically changing an element's position:

Code Sample:

DynamicHtml/Demos/Slider2.html

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="UTF-8">
<title>Slider</title>
<link href="slider2.css" rel="stylesheet" type="text/css">
<script type="text/javascript" src="../../lib.js"></script>
<script type="text/javascript">
  var timer, timesUp;
  function resetTimer(){
    var slider = document.getElementById("divSlider");
    timesUp = true;
    slider.style.left = "1px";
    clearInterval(timer);
  }

  function decrementTimer(){
    var slider = document.getElementById("divSlider");
    var curLeft = parseInt( getComputedStyle(slider,"left") );
    var curWidth = parseInt( getComputedStyle(slider,"width") );
    timesUp = false;
    if (curLeft < 198 - curWidth) {
      slider.style.left = curLeft + 2 + "px";
    } else {
      alert("Time's up!");
      resetTimer();
    }
  }

  observeEvent(window,"load",function() {
    var btnStart = document.getElementById("btnStart");
    resetTimer();
    observeEvent(btnStart,"click",function() {
      timer = setInterval(decrementTimer, 100);
    });
  });
</script>
```

```
</head>
<body>
<div id="divSliderBG">
  <div id="divSlider"></div>
</div>
<button id="btnStart">Start Timer</button>
</body>
</html>
```

Lesson 1, Activity 14: Changing the Math Quiz Timer to a Slider

Duration: 15 to 25 minutes.

In this exercise, you will modify the Math Quiz so that the timer is a slider rather than a count down. The result will look like this:

Category:	Addition ▾
Question:	5 + 4 ▾
Answer:	<input type="text"/> <input type="button" value="Check Answer"/>
Timer:	<div style="width: 100%; height: 20px; background-color: blue; position: relative;"><div style="position: absolute; left: 50%; width: 10px; height: 10px; background-color: red; transform: translate(-50%, -50%);"></div></div>
Good luck!	

Question	Your answer	Correct answer
----------	-------------	----------------

1. Open [DynamicHtml/Exercises/MathQuizSlider.html](#) for editing.
2. Notice that the timer on the page has been changed from an `input` element to two `divs`.

```
<tr id="timerRow">
  <td>Timer:</td>
  <td>
    <div id="divSliderBG">
      <div id="divSlider"></div>
    </div>
  </td>
</tr>
```

3. Also, all references to the `timeLeft` `input` have been removed and the code within the `decrementTimer()` function has been commented out.
4. Rewrite the `decrementTimer()` function to use a slider like the one shown in the screenshot above.
5. Test your solution in a browser.

Solution:

[DynamicHtml/Solutions/MathQuizSlider.html](#)

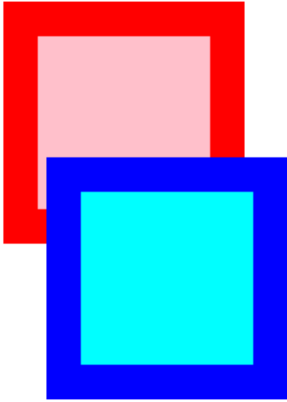
```
---- C O D E   O M I T T E D ----

function decrementTimer() {
  var slider = document.getElementById("divSlider");
  var curLeft = parseInt( getComputedStyle(slider,"left") );
  var curWidth = parseInt( getComputedStyle(slider,"width") );
  if (curLeft < 198 - curWidth) {
    slider.style.left = curLeft + 2 + "px";
  } else {
    resetTimer(timePerQuestion);
    msg("Time's up! The answer is " + getAnswer() + ".", "red");
    removeOption();
  }
}
```

```
}  
---- C O D E   O M I T T E D ----
```

Lesson 1, Activity 15: Changing the Z-Index

The z-index value of an element indicates its relative position in the "stack" of elements the page. Elements with higher z-index values sit on top of elements with lower values. You can think of a stack of papers thrown on a table. The ones at the top have a higher z-index than the ones on the bottom. In the screenshot below, the box on the bottom has a higher z-index that the box on top:



The z-index of an object can be changed by modifying the `zIndex` property of the element's `style` property. The following example demonstrates this:

Code Sample:

[DynamicHtml/Demos/Zindex.html](#)

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="UTF-8">
<title>zIndex</title>
<link href="zindex.css" rel="stylesheet" type="text/css">
<script type="text/javascript" src="../../lib.js"></script>
<script type="text/javascript">
    var z = 0;
    function changeZ(elem){
        z += 10;
        elem.style.zIndex = z;
        elem.innerHTML = "z: " + z;
    }

    observeEvent(window,"load",function() {
        var divRed = document.getElementById("divRed");
        var divBlue = document.getElementById("divBlue");
        observeEvent(divRed,"click",function() {
            changeZ(divRed);
        });
        observeEvent(divBlue,"click",function() {
            changeZ(divBlue);
        });
    });
</script>
</head>
<body>
```



```
<div id="divRed"></div>
<div id="divBlue"></div>
</body>
</html>
```

The variable `z` always holds the highest z-index. The function `changeZ()` simply adds 10 to `z` and assigns the resulting value to the `zIndex` property of the passed in object.

Lesson 1, Activity 16: A Note on JavaScript Frameworks

In the Demos folder of this lesson, you will find a Pong.html file that uses the features covered in this lesson to create the classic Pong game. Although the code is somewhat complex, with some study you should be able to understand everything in the file.

There are many JavaScript frameworks, such as [YUI](#), [jQuery](#), and [script.aculo.us](#) that make it much easier to write complicated Dynamic HTML applications that include drag-and-drop features and elements moving around the page. These frameworks use the concepts covered in this lesson, but save you a lot of the coding. If you are writing large applications requiring a lot of Dynamic HTML, you should consider using one of these frameworks.

Lesson 1, Activity 17: **JavaScript style Properties**

Each CSS property has a corresponding property of the JavaScript `style` object:

- If the CSS property is a simple word (e.g, `color`) then the JavaScript property is the same (e.g, `style.color`).
- If the CSS property has a dash in it (e.g, `background-color`) then the JavaScript property uses lower camel case (e.g, `style.backgroundColor`).