6.1 Introduction

Def: A *dynamic HTML document* is one whose tag attributes, tag contents, or element style properties can be changed after the document has been and is still being displayed by a browser

6.2 Positioning Elements

- CSS-P was released by W3C in 1997

- Completely supported by IE9, FX3, and Chrome

- The position of any element is dictated by the three style properties: `position`, `left`, and `top`

  - The three possible values of position are `absolute`, `relative`, and `static`

- *Absolute Positioning*

```
<p style = "position: absolute; left: 50px;
             top: 100px;">
```

→ SHOW `absPos.html`

6.2 Element Positioning (continued)

- If an element is nested inside another element and is absolutely positioned, the top and left properties are relative to the enclosing element

→ SHOW `absPos2.html`

- *Relative Positioning*

  - If no `top` and `left` properties are specified, the element is placed exactly where it would have been placed if no `position` property were given

    - But it can be moved later

→ SHOW `relPos.html`

- *Static Positioning*

  - The default value if `position` is not specified

6.3 Moving Elements

- If `position` is set to either `absolute` or `relative`, the element can be moved after it is displayed

  - Just change the `top` and `left` property values with a script

→SHOW `mover.html` & `mover.js`

6.4 Element Visibility

- The `visibility` property of an element controls whether it is displayed

  - The values are `visible` and `hidden`

```
if (dom.visibility == "visible"
   dom.visibility = "hidden";
else
   dom.visibility = "visible";
```

→ SHOW `showHide.html` & `showHide.js`

6.5 Changing Colors and Fonts

- Background color is controlled by the `backgroundColor` property

- Foreground color is controlled by the `color` property

```
Background color:
  <input type = "text" size = "10"
         name = "background"
         onchange = "setColor('background',
                                 this.value)">
```

→ SHOW `dynColors.html` & `dynColors.js`

6.5 Dynamic Colors and Fonts (continued)

- *Changing fonts*

  - We can change the font properties of any element that contains text by using the `mouseover` and `mouseout` events to trigger a script that makes the changes

```
onmouseover = "this.style.color = 'blue';
       this.style.font = 'italic 16pt Times';"
onmouseout = "this.style.color = 'black';
       this.style.font = 'normal 16pt Times';”
```

  - *JavaScript property names*:
    - For CSS attributes w/o hyphens – same
    - For CSS attributes w/hyphens – delete hyphen and capitalize the next letter – `font-size` -> `fontSize`

→SHOW `dynFont`

6.6 Dynamic Content

- The content of an HTML element is addressed with the `value` property of its associated JavaScript object
  - Example: a help box for a form

→ SHOW `dynValue.html & dynValue.js`

6.7 Stacking Elements

- The `z-index` attribute determines which element is in front and which are covered by the front element

- The JavaScript property associated with the `z-index` attribute is `zIndex`

- `z-index` can be changed dynamically (by changing `zIndex`)

- An image element can have an `onclick` attribute, so images can be clicked to trigger event handlers

- Anchors can also trigger event handlers when they are clicked

  - The `href` attribute can be set to call a JavaScript function by assigning it the call, with `'JAVASCRIPT'` attached to the call code

```
<a href = "JAVASCRIPT:fun()">
```

6.7. Stacking Elements (continued)

- To change stacking order, the handler function must change the `zIndex` property value of the element

- A call to the function from an element sets the `zIndex` value of the new top element to 10 and the `zIndex` value of the old top element to 0

  - It also sets the current top to the new top

→ SHOW `stacking.html` & `stacking.js`

6.8 Locating the Mouse Cursor

- The coordinates of the element that caused an event are available in the `clientX` and `clientY` properties of the `event` object

  - These are relative to upper left corner of the browser display window

  - `screenX` and `screenY` are relative to the upper left corner of the whole client screen

6.8 Locating the Mouse Cursor (continued)

- If we want to locate the mouse cursor when the mouse button is clicked, we can use the `click` event

→ SHOW `where.html` & `where.js`

6.9 Reacting to a Mouse Click

- A mouse click can be used to trigger an action, no matter where the mouse cursor is in the display

→ SHOW `anywhere.html` & `anywhere.js`

6.10 Slow Movement of Elements

- To animate an element, it must be moved by small amounts, many times, in rapid succession

- JavaScript has two ways to do this, but we cover just one:

```
setTimeout("fun()", n)
```

6.10 Slow Movement of Elements (continued)

- *Example*: move a text element from its initial position (100, 100) to a new position (300, 300)

  - Use the `onload` attribute of the `body` element to initialize the position of the element

  - Use a move function to change the `top` and `left` attributes by one pixel in the direction of the destination

  - *A problem*: coordinate properties are stored as strings, which include the units (`"150px"`)

  - *Another problem:* We need to use some HTML special characters (‘`<`‘ and ‘`--`’)

    1. XML parsers may remove all comments
    2. Put the script in a `CDATA` section
    3. Put JavaScript in separate file

    - These are problems of validation only

    - IE9, FX3, and Chrome deal correctly with comments

6.10 Slow Movement of Elements (continued)

→ SHOW `moveText.html` & `moveTextfuns.js`

6.11 Dragging and Dropping an Element

- We can use `mousedown`, `mousemove`, and `mouseup` events to grab, drag, and drop
- We know how to move an element - just change its `left` and `top` properties
- *Example*: magnetic poetry
  - The DOM 2 event model is required (the `Event` object and its property, `currentTarget`)
  - We use both DOM 0 and DOM 2 models (DOM 0 to call the `mousedown` handler, `grabber`)

6.11 Dragging and Dropping an Element

- *Drag and drop requires three processes*:

1. Get the dom of the element to be moved when the mouse button is pressed down (`onmousedown`) while the mouse cursor is over the element to be moved

   - We can get the DOM of the element on which an event occurred with the `currentTarget` property of the event object

2. Move the element by changing its `top` and `left` properties of the element as the mouse cursor is moved (`onmousemove`)

   - Compute the distance of each move as the difference between the current position (the `left` and `top` values) and the mouse click position (`clientX` and `clientY`)

3. Dropping the element when the mouse button is released by undefining the dom used to carry out the move

→ SHOW `dragNDrop.html` & `dragNDrop.js`