|  |
| --- |
| [[https://myetudes.org/etudes-melete-tool/images/printer.png](https://myetudes.org/portal/tool/4c4d3792-8b10-40ce-8016-d7a5ac569a1c/print_module.jsf?printModuleId=1436385322) Send to Printer](https://myetudes.org/portal/tool/4c4d3792-8b10-40ce-8016-d7a5ac569a1c/print_module.jsf?printModuleId=1436385322) | [Close Window](https://myetudes.org/portal/tool/4c4d3792-8b10-40ce-8016-d7a5ac569a1c/print_module.jsf?printModuleId=1436385322) |
|  | |
| 11. Scripting Style  CSS and JavaScript Animations, Transitions  11.1. Separation of Layers  *Copyright (c) 2014, Rula Khayrallah*  So far our web pages have included content, provided by the HTML layer, and a certain behavior, implemented in JavaScript.  It’s about time we add some 'style' with the CSS layer.  CSS stands for Cascading Style Sheets.  Styles define the presentation of a web page, how HTML elements will be displayed.  There are several ways to associate styles with an HTML document.  In this course, we’ll stick to our recommended approach of separation of layers.  Just as we separated our JavaScript code from HTML documents, we’ll separate our CSS from both JavaScript code and from HTML.  We’ll place CSS in a separate external file that contains only CSS code.  That file will be saved with a 'css' extension and will be referenced in our HTML using the <link> tag. An advantage of this approach is that we can then change the appearance of several web pages by editing a single css file.  Let’s add an external style sheet to our classexample.html document:  <!DOCTYPE html>  <html>          <head>                   <meta charset = "utf-8">                  <title>JavaScript for Programmers</title>  **<link rel = "stylesheet" type = "text/css" href = "mystyle.css" media = "all">**          </head>          <body>                  <h1 class = "important">Layout Engines</h1>                  <p id = "mainidea" >                          <span id = "first" class = "info">The different browsers rely on layout engines. </span>                          <span id = "second" class = "important info">                                  The different layout engines implement the DOM standards to varying degrees of compliance.                          </span>                  </p>          </body>  </html>    **Note that the <link> tag must be placed inside the <head> element.**  The <link> tag has several attributes:  The **href** attribute specifies the location of our external css file.  The **rel** attribute specifies the relationship between the HTML document and the linked document:  in our case it’s the stylesheet.  The **type** attribute describes the type of the linked document:  it is "text/css".  Finally the **media** attribute specifies the media or device that the stylesheet is to be associated with.   Valid media include "all", "handheld", "print" and "screen".  11.2. Cascading Style Sheets  *Copyright (c) 2014, Rula Khayrallah*  We have seen how to associate an external stylesheet with an HTML document but we have not seen what goes in that stylesheet.  CSS is a powerful language with a simple syntax.  The in-depth study of CSS is beyond the scope of this course.  We’ll just demonstrate the basics with some examples.  A CSS stylesheet includes a set of **rules**.  Each rule consists of one or more **selectors**, and a **declaration block**.  selector {        declaration block  }  The selector refers to the HTML element we want to style.  The declaration block may contain several declarations.  Each declaration consists of a property and a value.  The property is the style attribute we want to change. Each property has a value.  **Selecting elements by their tag:**   tag {...  To define a style on <p> elements, we use  p as our selector.  We can include comments between /\* and \*/.  We can include more than one declaration in each rule, we separate them with a semicolon.  p {  /\* the selector "p" matches all <p> elements \*/  color: blue;  font-size: 20px;  }  To define a background color for the whole document, we write the following rule:  body { /\* the selector "body" matches the document body \*/        background-color: #c0e4fe;  }  **Selecting elements by their id:**  #id {...  **We can also refer to an element** **by its id**.**We just prefix the id with #.**  So to style the element with the id "second", we use #second as our selector.  #second {  /\* the selector matches the element with id "second" \*/        color: red;  }  The red color for the more specific id rule here will override the blue property for the more general p rule.  The font-size will be used from the p rule.  **Selecting elements by their class:**  .class {...  Finally,**we can also refer to several elements by their class**.  **We prefix the class name with a dot.**  So to style all the element with class =  "important", we use .important (dot important) as our selector.  .important { /\* the selector matches all elements with class "important" \*/  font-weight: bold;  }  We are now ready to create our stylesheet, mystyle.css.  body {        background-color: #c0e4fe;  }  p { /\* the selector "p" matches all < p > elements \*/        color: blue;        font-size: 20px;  }  #second {  /\* the selector matches the element with id "second" \*/        color: red;  }  .important {        font-weight: bold;  }  We see the style changes to the web page when we open the modified classexample.html in Firefox.  Just make sure you add the <link> tag to the html source first.  11.3. Changing Styles with JavaScript  *Copyright (c) 2014, Rula Khayrallah*  Now that we’ve seen how styles are defined in CSS, we’ll see how to change these styles from JavaScript.  **The recommended way to change the style of a given element is to change the value of its class attribute.** **In the DOM, that attribute is denoted by className**.  We’ll see how to do that in the following simple example.  Let’s associate our classexample.html source file with a JavaScript program, styleselect.js.  <!DOCTYPE html>  <html>          <head>                  <meta charset = "utf-8">                  <title>JavaScript for Programmers</title>                  <link rel = "stylesheet" type = "text/css" href = "mystyle.css" media = "all">          </head>          <body>                  <h1 class = "important">Layout Engines</h1>                  <p id = "mainidea" >                          <span id = "first" class = "info">The different browsers rely on layout engines. </span>                          <span id = "second" class = "important info">                                  The different layout engines implement the DOM standards to varying degrees of compliance.                          </span>                  </p>  **<script defer src="../scripts/styleselect.js"></script>**          </body>  </html>  Let’s also modify our style sheet, mystyle.css, to add a new rule for class= "selected" as shown below.  body {          background-color: #c0e4fe;  }  p { /\* the selector "p" matches all < p > elements \*/          color: blue;          font-size: 20px;  }  #second {  /\* the selector matches the element with id "second" \*/          color: red;  }  .important {          font-weight: bold;  }  **.selected {**  **color: green;**  **}**  Now we create our JavaScript program, styleselect.js program as follows:  styleselect.js  function green(event) {      // if the className does not include selected      if (event.target.className.indexOf('selected') < 0) {           event.target.className = event.target.className + ' selected';      }  };    document.body.addEventListener('mouseover', green, false);  Here we have an event listener registered on the body of the document for a mouseover event. **So whenever our mouse goes over a target element, the function green adds "selected"  to its class attribute if it is not already there.**   We do that because we don’t want to overwrite its existing class, we just want to add "selected" to it.  We also don't want to add "selected" multiple times.  Remember that className specifies **multiple class names separated by a space**.  **Once the class of the given element is changed to selected, its text color is changed to green as per the CSS stylesheet.**  Note again that the more specific CSS rules override the more general rules.  So the class selector .selected is more specific than the tag selector p.  Hence a p element with a class = "selected" will be green not blue.  On the other hand the id selector #second is  more specific than the class selector .selected so the element "second" will still be red even though it belongs to the selected class too.  11.4. Hide and Show  *Copyright (c) 2014, Rula Khayrallah*  Sometimes, we need to hide and show elements selectively on a web page.   Let’s see how we can do that from within our JavaScript code.  There are two main methods to hide elements with CSS.  We can set the **display property**of an element to "none" or the **visibility property** to "hidden".  The two methods produce difference results.  Let’s illustrate that with an example.  First let’s create the following HTML source document peekaboo.html:  peekaboo.html  <!DOCTYPE html>  <html>        <head>              <meta charset = "utf-8">              <title>JavaScript for Programmers</title>  **<link rel = "stylesheet" type = "text/ css" href = "peekaboo.css" media = "all">**        </head>        <body>              <h2>Peekaboo Demo</h2>              <p>Press on a button to change the visibility or display.</p>              <img id = "face" src = "hello.gif" class = "show" alt = "Hello">              <p>                    <input id = "visibility" type = "button" value = "Visibility">                    <input id = "display" type = "button" value = "Display">              </p>              <script defer src="../scripts/peekaboo.js"></script>        </body>  </html>  Here we have one image that we’ll hide and show when the user clicks on one of two buttons.  Here’s the corresponding syle sheet peekaboo.css:  peekaboo.css  body {  background-color: #e6e6ff;  }    .show {  visibility: visible;  display: inline;  }    .hide {  **visibility: hidden;**  }    .disappear {  **display: none;**  }  In the associated style sheet peekaboo.css, we define rules for the show class, for the hide class and for the disappear class.  For the hide class, we hide the element using the visibility property:  this will cause the element to disappear, but it will still take up space on the web page.  For the disappear class, we hide the element using the display property: this  will cause the element to disappear and the place that was taken by the element on the page will be used by the elements coming after it.  Finally, we write our Javascript program peekaboo.js to tie everything together.  We need event listeners so that when the buttons are clicked, the appropriate action is taken.  We also define two functions that will toggle the class of the image element between show and hide (for the visibility button) and between show and disappear (for the display button).  peekaboo.js:  function toggleVisibility(event) {      if (document.getElementById('face').className === 'show') {          document.getElementById('face').className = 'hide';      } else if (document.getElementById('face').className === 'hide') {          document.getElementById('face').className = 'show';      }  };  function toggleDisplay(event) {      if (document.getElementById('face').className === 'show') {          document.getElementById('face').className = 'disappear';      } else if (document.getElementById('face').className === 'disappear') {          document.getElementById('face').className = 'show';      }  };  document.getElementById('visibility').addEventListener('click', toggleVisibility, false);  document.getElementById('display').addEventListener('click', toggleDisplay, false);  When we open the source document peekaboo.html in Firefox, we initially get the following:  When we press the visibility button, the function toggleVisibilty is called and the className corresponding to the image is change to 'hidden'.  This causes the image to disappear from the page as shown below.  However the space that was initially allocated to the image is still taken.  Now let’s press the Visibility button again to restore the image.  Then let’s click on the Display button:  the function toggleDisplay is called now and the className corresponding to the image is now changed to 'disappear'.  This causes the image to disappear form the page as shown below.  Note that this time the buttons have moved up to occupy the space that was initially allocated to the image.  11.5. Transition Example  *Copyright (c) 2014, Rula Khayrallah*  We have seen how to use the timer functions available in JavaScript to create animated effects.  CSS3 now defines a way to specify transitions and animations in stylesheets.  We can add a transition effect when changing from one style to another.  **We specify the transition for the class we are transitioning into**.  We also specify **each CSS property** we want to add an effect to and the **duration of the transition**.  CSS transitions are supported in the current browsers; however Safari requires the prefix -webkit-.  To see how transitions can be controlled from JavaScript, we’ll first create a web page that implements a zooming transition effect, tree.html.  <!DOCTYPE html>  <html>          <head>                  <meta charset = "utf-8">                  <title>JavaScript for Programmers</title>  **<link rel = "stylesheet" type = "text/ css" href = "transitions.css" media = "all">**          </head>          <body>              <h2>Transition Demo</h2>              <p>We create the zooming transition in CSS and trigger it from JavaScript.</p>              <p><input id = "mybutton" type="button" value="Press Here"></p>  **<img id="tree" src="tree.png" alt="tree" class = "small">**  **<script defer src="../scripts/tree.js"></script>**          </body>  </html>  And the corresponding stylesheet defines styles for the "small" and "large" classes with 5 seconds transitions on width and height as follows:  transitions.css  body {        background-color: #c0e4fe;  }  .small {        width: 100px;        height: 100px;  **transition: height 5s, width 5s**;   **/\* transition on height&width, duration 5 seconds**\*/        -webkit-transition: height 5s, width 5s; /\* Safari \*/  }  .large {        width: 300px;        height: 300px;  **transition: height 5s, width 5s;   /\* transition on height&width, duration 5 seconds \*/**        -webkit-transition: height 5s, width 5s; /\* Safari \*/  }  Finally we write our JavaScript program tree.js as follows:  function toggle(event) {      if (document.getElementById('tree').className === 'small'){          document.getElementById('tree').className = 'large';      }else {          document.getElementById('tree').className = 'small';      }  };  document.getElementById('mybutton').addEventListener('click', toggle, false);  The tree picture tree.png is available under Resources. You can see how the transition works by opening the tree.html file and pressing the button a couple of times.  And after you press the button:    11.6. Animation Example  *Copyright (c) 2014, Rula Khayrallah*  In our next example, we’ll create an animation in CSS and control it from JavaScript.  To create animations in CSS3, we specify two styles inside the **@keyframes rule** and the animation will gradually change from the current style (denoted by 'from') to the new style denoted by 'to'.  @keyframes **moving**{  **from** {              }  **to** {                          transform: translateX(800px) rotate(360deg);              }  }  Here the 'moving' animation will move the element selected 800 pixels along the horizontal X axis (translateX(800px)) and rotate it 360 degrees rotate(360deg).  To   bind the animation to a selector (such as the 'move' class), we must specify **a name and a duration.**  We can also specify other properties:  **infinite** will make the animation run forever, and  **alternate** will cause the animation to play in reverse on alternate cycles.  .move {  **animation: moving 8s infinite alternate;**  }  CSS animations are supported in the current browsers, however Safari and Chrome require the prefix -webkit-.  So in order to support Safari and Chrome, we update our CSS stylesheet to include the following:  .move{              animation: moving 8s infinite alternate;  **-webkit-animation: moving 8s infinite alternate;/\* Chrome and Safari \*/**  }  @keyframes moving{              from {              }              to {                          transform: translateX(800px) rotate(360deg);              }  }  **@-webkit-keyframes moving** {              from {              }              to {                          -webkit-transform: translateX(800px) rotate(360deg);              }  }  We then create a soccer.html document that will use our stylesheet to animate a soccer ball:  <!DOCTYPE html>  <html>                  <head>                                  <meta charset = "utf-8">                                  <title>JavaScript for Programmers</title>  **<link rel = "stylesheet" type = "text/css" href = "animate.css" media = "all">**                  </head>                  <body>                                  <h2>Animation Demo</h2>                                  <p>We create the animation in CSS and trigger it from JavaScript.</p>                                  <input id = "gobutton" type="button" value="GO">                                  <input id = "stopbutton" type="button" value="STOP">                                  <p>Click on the GO button to move the ball.</p>  **<img id="ball" src="ball.png" alt="ball">**                                  <**script defer src="../scripts/animate.js"></script>**                  </body>  </html>    We refine our stylesheet to include a background color as follows:  animate.css  body {              background-color:#98ff70;  }  .move{              animation: **moving** 8s infinite alternate;              -webkit-animation: **moving** 8s infinite alternate;/\* Chrome and Safari \*/  }  @keyframes **moving** {              from {              }              to {                          transform: translateX(800px) rotate(360deg);              }  }  @-webkit-keyframes **moving** {              from {              }              to {                          -webkit-transform: translateX(800px) rotate(360deg);              }  }  All we have left to do is create our JavaScript program to control the animation with user clicks.  We create animate.js as follows.  function move(event) {    // change the className to 'move' in order to move the ball      document.getElementById('ball').className = 'move';  }  function stop(event) {    // change the className to '' so that it is no longer 'move'      document.getElementById('ball').className = '';  }  document.getElementById('gobutton').addEventListener('click', move, false);  document.getElementById('stopbutton').addEventListener('click', stop, false);    The two listener functions are simply changing the class of the ball element.  Note that the file ball.png is available under Resources.  We can now open the file soccer.html start the animation by pressing on the Go button: |  |