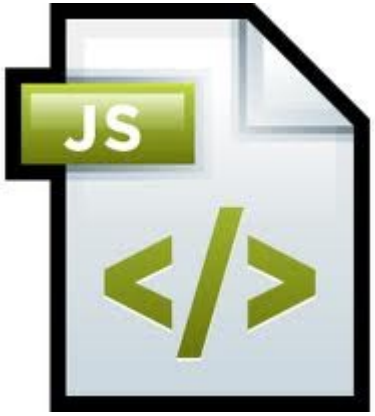


Scripting CSS

Browser events

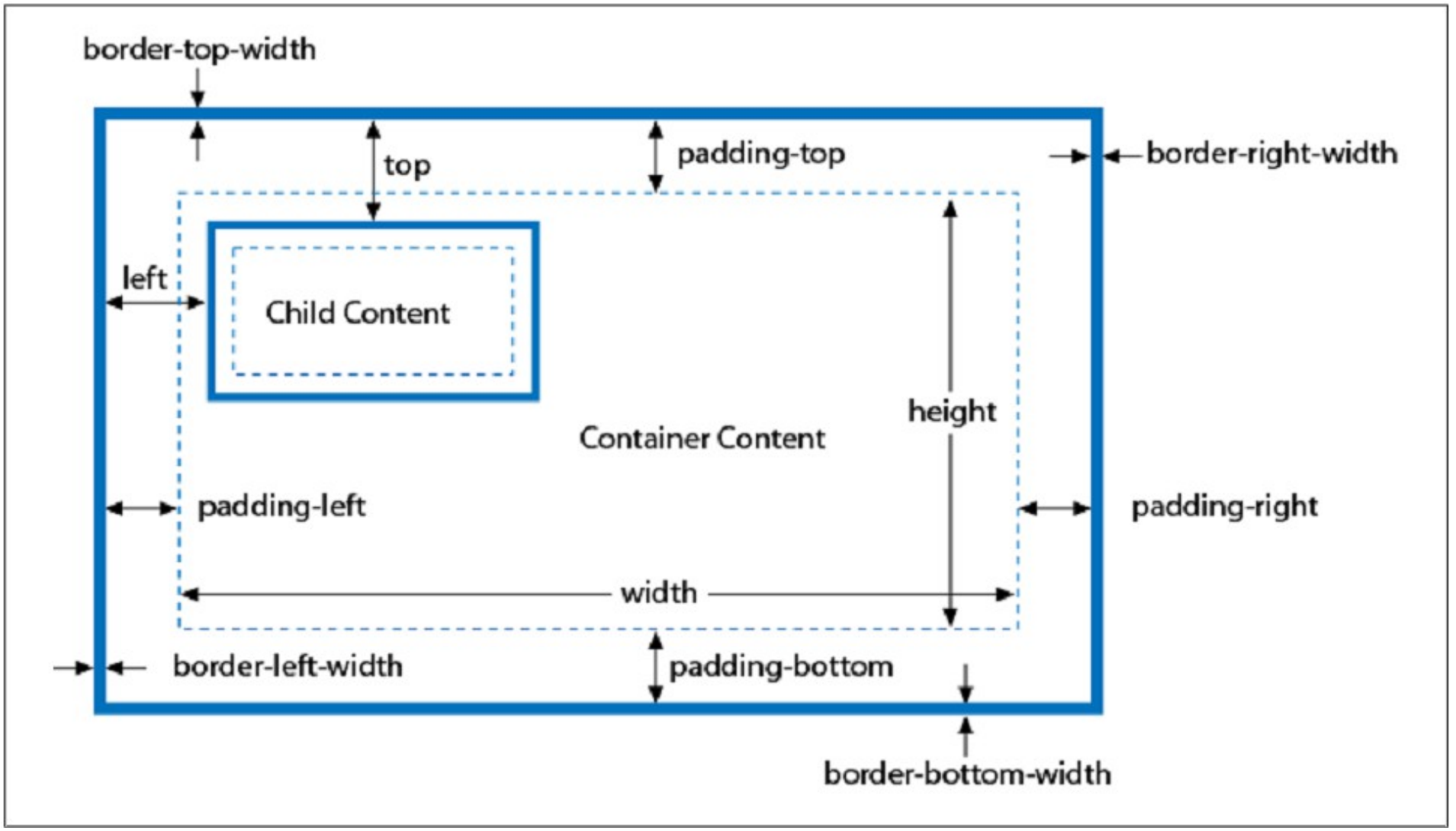
Own objects, properties and methods

Scripting CSS

- Scripted CSS enables a variety of interesting visual effects
  - You can create animated transitions where document content "slides in" from the side
  - Or create an expanding and collapsing outline list in which the user can control the amount of information that is displayed
- Important CSS style properties for scripting

Property	Description
`position`	Specifies the type of positioning applied to an element
`top, left`	Specify the position of the top and left edges of an element
`bottom, right`	Specify the position of the bottom and right edges of an element
`width, height`	Specify the size of an element
`z-index`	Specifies the "stacking order" of an element relative to any overlapping elements; defines a third dimension of element positioning
`display`	Specifies how and whether an element is displayed
`visibility`	Specifies whether an element is visible
`clip`	Defines a "clipping region" for an element; only portions of the element within this region are displayed
`overflow`	Specifies what to do if an element is bigger than the space allotted for it
`margin, border, padding`	Specify spacing and borders for an element.
`background`	Specifies the background color or image of an element.
`opacity`	Specifies how opaque (or translucent) an element is. This is a CSS3 property, supported by some browsers. A working alternative exists for IE.

The CSS Box Model and Positioning Details



Changing a style

- To script CSS we alter the style attribute of individual document elements
- The inline style property is a CSSStyleDeclaration object
- Note that many CSS style properties, such as font-size etc. contain hyphens in their names - which in JavaScript is interpreted as a minus sign!
- You can get individual inline styles for an element with the getComputedStyle() method

findchangecssrules.html

```
//Obtain an element
var e = document.getElementById("id");
// Obtain the computed style for a specific element
var size = parseInt(window.getComputedStyle(e,"").fontSize);
// To make the text of an element e big, bold and blue
e.style.font-size = "24pt"; // Syntax error!
e.style.fontSize = "24pt"; e.style.fontWeight = "bold"; e.style.color = "blue";
// To set the first element found with 'tagname' to a specific CSS class
e = document.getElementsByTagName('tagname')[0];
e.className = "CSSclassname";
```

Scripting stylesheets

- Permits access and modify possibilities of rules in a stylesheet
- There may be differences in implementation of CSS between different browsers
- A simple stylesheet to the right
- The example below searches for the "a:hover" rule within the style sheet, and if found, returns a reference to it

```
<style>
b{
     color: orange;
     font-size: 110%;
}

a:hover{
     background-color: yellow;
}

#test{
     border: 1px solid black;
}
</style>
```

```
var mysheet = document.styleSheets[0];
// firefox or IE?
var myrules =
     mysheet.cssRules? mysheet.cssRules : mysheet.rules;
for (var i=0; i<myrules.length; i++){
     if(myrules[i].selectorText.toLowerCase() == "a:hover"){
          //find "a:hover" rule
          var targetrule = myrules[i];
          alert("Found a:hover " + targetrule);
          break;
     }
}
```

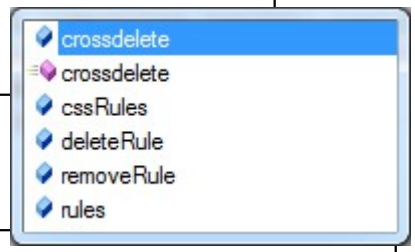
findchangecssrules.html

Change and delete stylesheets

- The first CSS rule ("b") contains two attributes. Let us change the color attribute from "orange" to "red"

findchangecssrules.html

```
var mysheet = document.styleSheets[0];
var firstrule = mysheet.cssRules? mysheet.cssRules[0] : mysheet.rules[0];
if (firstrule.style.color == "orange")
     firstrule.style.color = "red";
```



- Delete an entire CSS rule

```
var mysheet = document.styleSheets[0];
// firefox or IE?
var myrules = mysheet.cssRules? mysheet.cssRules : mysheet.rules;
mysheet.crossdelete = mysheet.deleteRule? mysheet.deleteRule : mysheet.removeRule;
for (i=0; i<myrules.length; i++){
     if(myrules[i].selectorText.toLowerCase().indexOf("#test") != -1){
          mysheet.crossdelete(i);
          i--; //decrement i by 1, since deleting a rule collapses the array by 1
     }
}
```

http://www.javascriptkit.com/dhtmltutors/externalcss3.shtml

Handling events

- Events are occurrences that the web browser will notify your program about
  - Most computer applications use an asynchronous event-driven programming model
- The ***event type*** or ***event name*** is a string that specifies what kind of event occurred (onload)
- The ***event target*** is the object on which the event occurred or with which the event is associated (window)
- An ***event handler*** or ***event listener*** is a function that handles or responds to an event (function)
- An ***event object*** is an object that is associated with a particular event and contains details about that event
  - The event object is passed in as argument to the event handler function, the ***MouseEvent*** for example
- ***Event propagation*** is the process by which the browser decides which objects to trigger event handlers on

Browser events

- Legacy Event Types
  - The events you'll use most often in your web apps are generally the ones that have been around the longest and are universally supported: events for dealing with the **mouse**, the **keyboard**, **HTML forms**, and the **Window object**
- DOM Level 3 Events
  - The specification has been under development by the W3C for about a decade. It standardizes many of the legacy events and adds some new ones
  - These new event types are not yet widely supported, but browser vendors are expected to implement them once the standard is final
- HTML5 Events
  - HTML5 and related standards define a host of new APIs. Many of these APIs define events
  - Some of these events are ready to be used now. Others are not yet widely implemented and are not documented in any detail
- Touch based events for mobile phones as gestures etc.

Legacy/DOM events

- Every element on a web page has certain events which can trigger a JavaScript
  - For example, we can use the **onclick** event of a button element to indicate that a function will run when a user clicks on the button
  - We define the events in the HTML tags
- Examples of events which can be handled
  - A mouse click
  - A web page or an image loading
  - Mousing over a hot spot on the web page
  - Selecting an input field in an HTML form
  - Submitting an HTML form
  - A keystroke
- Note: Events are normally used in combination with functions, and the function will not be executed before the event occurs!

HTML4 standard event attributes 1

- HTML4 added the ability to let global events trigger actions in a browser, like starting a JavaScript when a user clicks on an element
- http://www.w3schools.com/tags/ref_eventattributes.asp

Mouse Events

Valid in all elements except base, bdo, br, frame, frameset, head, html, iframe, meta, param, script, style, and title.

Attribute	Value	Description
onclick	*script*	Script to be run on a mouse click
ondblclick	*script*	Script to be run on a mouse double-click
onmousedown	*script*	Script to be run when mouse button is pressed
onmousemove	*script*	Script to be run when mouse pointer moves
onmouseout	*script*	Script to be run when mouse pointer moves out of an element
onmouseover	*script*	Script to be run when mouse pointer moves over an element
onmouseup	*script*	Script to be run when mouse button is released

HTML4 standard event attributes 2

<body> and <frameset> Events

The two attributes below can only be used in <body> or <frameset>:

Attribute	Value	Description
onload	*script*	Script to be run when a document load
onunload	*script*	Script to be run when a document unload

Form Events

The attributes below can be used in form elements:

Attribute	Value	Description
onblur	*script*	Script to be run when an element loses focus
onchange	*script*	Script to be run when an element changes
onfocus	*script*	Script to be run when an element gets focus
onreset	*script*	Script to be run when a form is reset
onselect	*script*	Script to be run when an element is selected
onsubmit	*script*	Script to be run when a form is submitted

HTML4 standard event attributes 3

- HTML5 got an additional 50 global events
- http://www.w3schools.com/html5/html5_ref_eventattributes.asp

Image Events

The attribute below can be used with the img element:

Attribute	Value	Description
onabort	*script*	Script to be run when loading of an image is interrupted

Keyboard Events

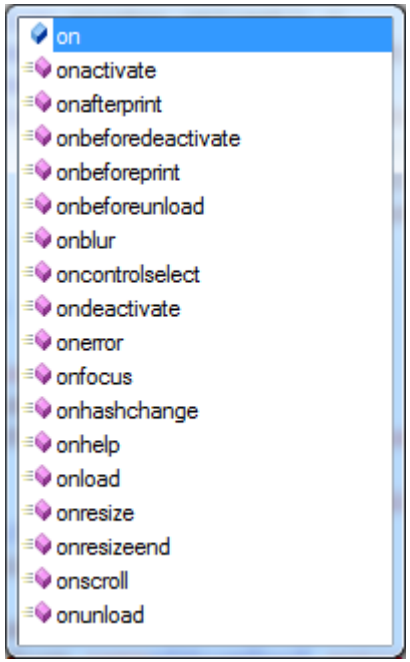
Valid in all elements except base, bdo, br, frame, frameset, head, html, iframe, meta, param, script, style, and title.

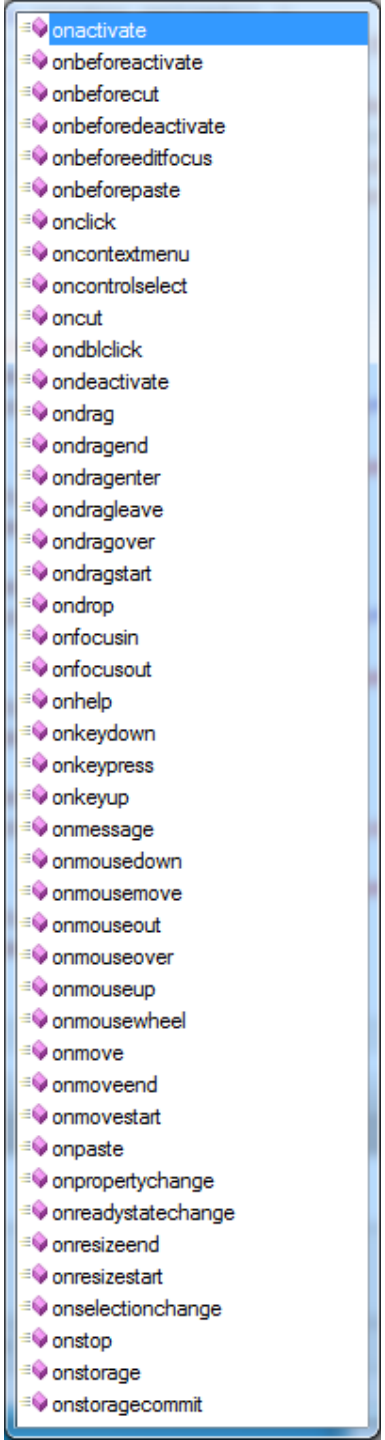
Attribute	Value	Description
onkeydown	*script*	Script to be run when a key is pressed
onkeypress	*script*	Script to be run when a key is pressed and released
onkeyup	*script*	Script to be run when a key is released

window vs. document events

- window.on*
- document.on*

DOMContentLoaded and readystatechange are alternatives to the load event: they are triggered sooner, when the document and its elements are ready to be manipulated, but before external resources are fully loaded.





onload and onunload

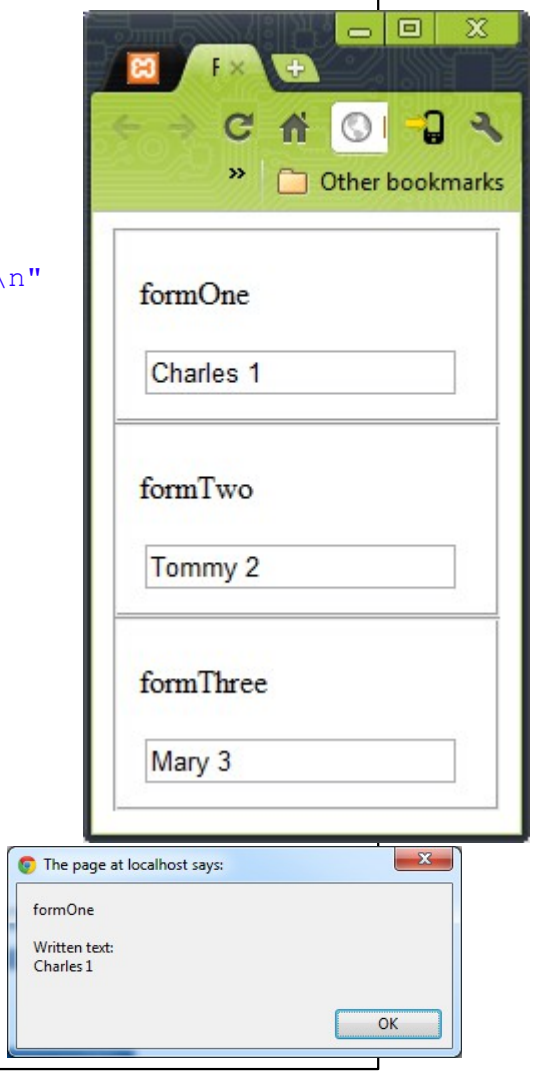
- The onload and onunload events are triggered when the user enters or leaves the page.
- The onload event is often used to check the visitor's browser type and browser version, and load the proper version of the web page based on the information.
- Both the onload and onunload events are also often used to deal with cookies that should be set when a user enters or leaves a page.
  - For example, you could have a popup asking for the user's name upon his first arrival to your page. The name is then stored in a cookie. Next time the visitor arrives at your page, you could have another popup saying something like: "Welcome John Doe!".

onload and forms array

- Forms can be accessed as an array

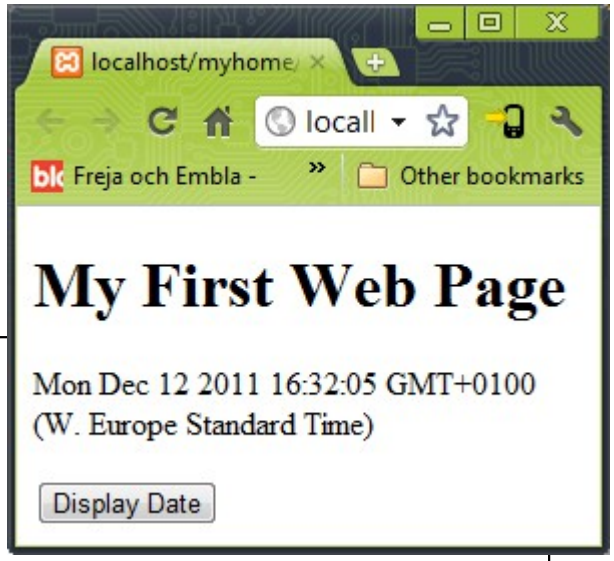
forms_array.html

```
<head>
<meta charset="iso-8859-1">
<title>Forms Array</title>
<script>
      function window_onload() {
            var numberForms = document.forms.length; //number of forms
            for (var formIndex = 0; formIndex < numberForms; formIndex++) {
                  //returns the value (name="...") for every form
                  alert(document.forms[formIndex].name + "\n\nWritten text:\n"
                              + document.forms[formIndex].inp.value);
            }
      }
</script>
</head>
<body onload="window_onload()">
      <fieldset><form name="formOne" action="#">
                  <p>formOne</p>
                  <input type="text" value="Charles 1" name="inp" />
            </form></fieldset>

      <fieldset><form name="formTwo" action="#">
                  <p>formTwo</p>
                  <input type="text" value="Tommy 2" name="inp" />
            </form></fieldset>

      <fieldset><form name="formThree" action="#">
                  <p>formThree</p>
                  <input type="text" value="Mary 3" name="inp" />
            </form></fieldset>
</body>
```



onclick

- Use document.getElementById and innerHTML to update the paragraph




```
<!doctype html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>onclick</title>
<meta name="" content="">
<link rel="stylesheet" href="css/styles.css">
<!--[if lt IE 9]>
     <script src="http://html5shiv.googlecode.com/svn/trunk/html5.js">
     </script>
<![endif]-->
<script>
     function displayDate() {
          document.getElementById("demo").innerHTML = Date();
     }
</script>
</head>
<body>
     <h1>My First Web Page</h1>
     <p id="demo"></p>
     <button type="button" onclick="displayDate()">Display Date</button>
</body>
</html>
```

onclick.html

onfocus, onblur and onchange

- The onfocus, onblur and onchange events are often used in combination with validation of form fields

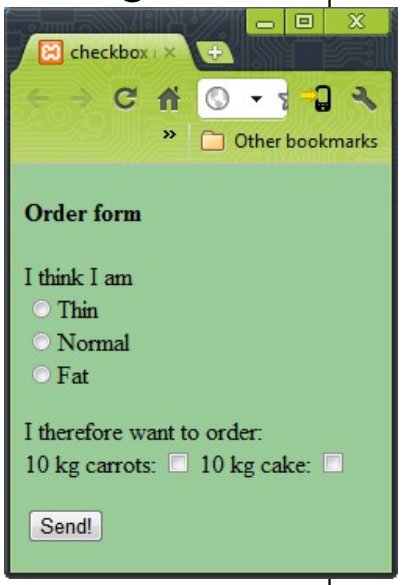
texteventhandlers.html

```
<head>
<title>Test event handlers</title>
<style>
span {
      font-family: courier;
}
</style>
</head>
<body>
      <h3>Event handlers:</h3>
      <p><span>onselect:</span><br />
      <!-- if user select the text -->
      <input type="text" value="text1"
            onselect="alert('onselect:\nYou selected the text')" /></p>
      <p><span>onfocus:</span><br />
      <!-- if user clicks with the mouse in the text or use tab to enter -->
      <input type="text" value="text2"
            onfocus="alert('onfocus:\nYou now have focus on the text')" /></p>
      <p><span>onblur:</span><br />
      <!-- when the user have had focus on the textfield and leave it -->
      <input type="text" value="text3"
            onblur="alert('onblur:\nYou left the text')" /></p>
      <p><span>onkeydown:</span><br />
      <!-- if the user press down a keyboard button when in the textfield -->
      <input type="text" value="text4"
            onkeypress="alert('onkeydown:\nYou pressed a keyboard button')" /></p>
</body>
```

onsubmit and checkbox

- The onsubmit event is used to validate ALL form fields before submitting it

checkbox_radio_event.html



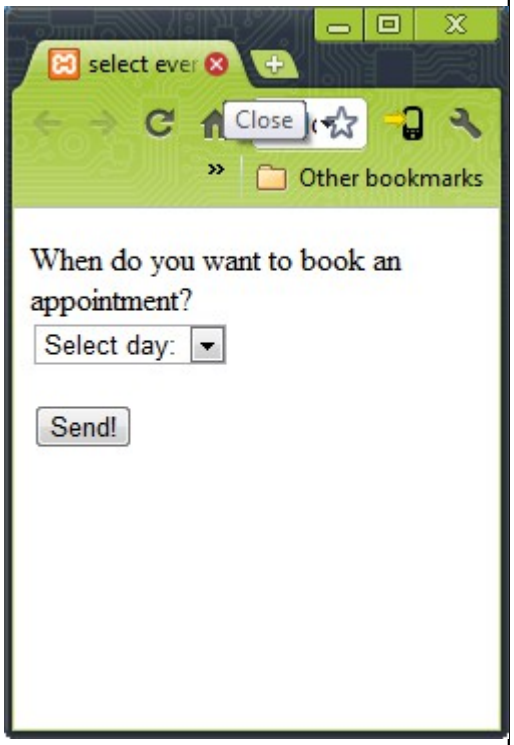

```
<head>
<meta charset="utf-8">
<title>checkbox radio event</title>
<script>
function validate_form() {
	var valid = true;
	if ((document.bestall.typ[0].checked == false) && (document.bestall.typ[1].checked == false)
			&& (document.bestall.typ[2].checked == false)) {
		alert("You must decide!\n(Thin, normal or fat)");
		document.bestall.typ[1].focus();
		valid = false;
	}
	if ((document.bestall.morot.checked == true) && (document.bestall.cake.checked == true)) {
		alert("Carrots and cake cannot be delivered together!");
		valid = false;
	}
	return valid;
}
</script>
</head>
<body style="background-color: #99cc99;">
	<h4>Order form</h4>
<!-- The return allows us to return the value true or false from our function to the browser, where true means
"carry on and send the form to the server", and false means "don't send the form". This means that we can prevent
the form from being sent if the user hasn't filled it in properly. -->
	<form name="bestall" method="get" action="#" onsubmit="return validate_form();">
		<p>I think I am<br /> <input type="radio" name="typ" value="smal" />Thin<br />
			<input type="radio" name="typ" value="norm" />Normal<br />
			<input type="radio" name="typ" value="fet" />Fat</p>

		<p>I therefore want to order: <br /> 10 kg carrots:
			<input type="checkbox" name="morot" value="Ja" /> 10 kg cake:
			<input type="checkbox" name="cake" value="Ja" /></p>
		<p><input type="submit" value="Send!" /></p>
	</form>
</body>
```

onsubmit and select

- The onsubmit event is used to validate ALL form fields before submitting it

select_eventhandler.html

```
<head>
<meta charset="utf-8">
<title>select eventhandler</title>
<script>
	function approved() {
		var valid = true;
		var valdDag = document.form1.dropmeny.selectedIndex;
		if (valdDag == 0) {
			alert("Kindly select a day!");
			valid = false;
		} else {
			alert("You have booked "
				+ document.form1.dropmeny.options[valdDag].text
				+ "\n\nvalue in \<option\> is: "
				+ document.form1.dropmeny.options[valdDag].value);
		}
		return valid;
	}
</script>
</head>
<body>
	<form name="form1" action="#" onsubmit="return approved()">
		<p>When do you want to book an appointment? <br />
			<select name="dropmeny" size="1">
				<option value="0">Select day:</option>
				<option value="mon">Monday</option>
				<option value="tue">Tuesday</option>
				<option value="wed">Wednesday</option>
				<option value="thu">Thursday</option>
				<option value="fri">Friday</option>
				<option value="sat">Saturday</option>
				<option value="sun">Sunday</option>
			</select></p>
		<input type="submit" value="Send!" />
	</form>
</body>
```



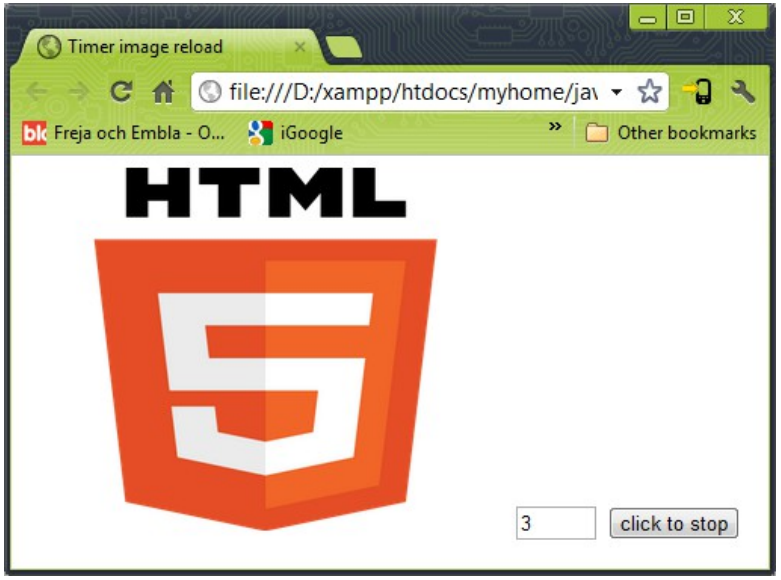
Timer example

- Images can be accessed directly by using the name attribute or the objects index (document.object[].) array

```
<!doctype html>
<html>
<head>
<title>Timer image reload</title>
<meta charset="iso-8859-1">
<script>
	var refreshrate = 5;
	var x = refreshrate;
	var timerID;

	function stopClock(){
		clearTimeout(timerID);
		document.form0.button.value =
			"dblclick to restart";
	}

	function startClock(){
		document.form0.button.value = "click to stop";
		x = x - 1;
		document.form0.clock.value = x;
		if (x < 1)
			reload();
		timerID = setTimeout("startClock()", 1000);
	}

	function reload() {
		var img = timerID % 2 ? "image1.jpg" : "image2.png";
		document.pic.src = img;
		//document.images[0].src = img;
		x = refreshrate;
		document.forms[0].clock.value = x;
	}
</script>
</head>
```

```
<body> <!--onload="startClock()"-->
	<form action="#" name="form0">
		<img src="image1.jpg" name="pic" alt="my picture">
		<input type="text" name="clock" size="3" value="">
		<input type="button" name="button" value=""
		    onclick="stopClock()" ondblclick="startClock()">
	</form>

<script>
	window.onload = startClock;
</script>
</body>
</html>
```

timer_imagereload.html

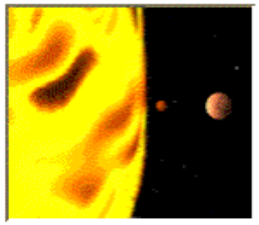


onmouseover

- The onmouseover event can be used to trigger a function when the user mouses over an HTML element:

```
<!doctype html>
<html>
<head>
<script>
        function writeText(txt){
                document.getElementById("desc").innerHTML=txt;
        }
</script>
</head>

<body>
<img src ="planets.gif" width ="145" height ="126" alt="Planets" usemap="#planetmap" />

<map name="planetmap">
<area shape ="rect" coords ="0,0,82,126"
        onmouseover="writeText('The Sun and the gas giant planets like Jupiter are by far the largest objects in our Solar System.')"
                href ="sun.htm" target ="_blank" alt="Sun" />

<area shape ="circle" coords ="90,58,3"
        onmouseover="writeText('The planet Mercury is very difficult to study from the Earth because it is always so close to the Sun.')"
                href ="mercur.htm" target ="_blank" alt="Mercury" />

<area shape ="circle" coords ="124,58,8"
        onmouseover="writeText('Until the 1960s, Venus was often considered a twin sister to the Earth
                because Venus is the nearest planet to us, and because the two planets seem to share many characteristics.')"
                href ="venus.htm" target ="_blank" alt="Venus" />
</map>

<p id="desc">Mouse over the sun and the planets and see the different descriptions.</p>
</body>
</html>
```

onmousover.html

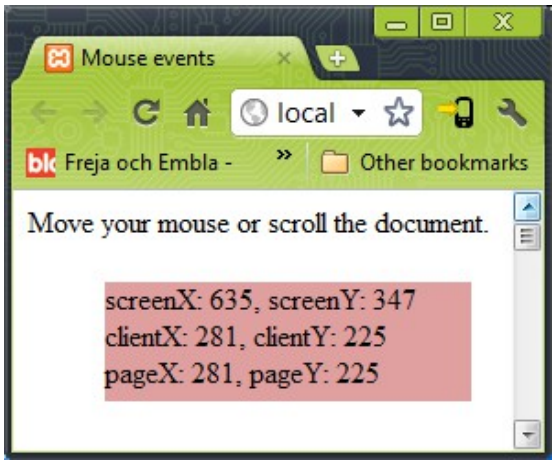


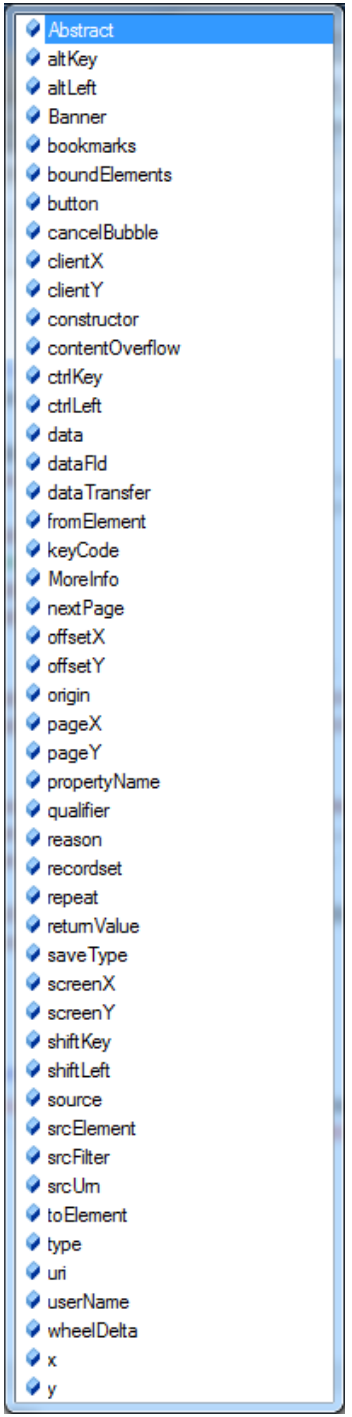
The Sun and the gas giant planets like Jupiter are by far the largest objects in our Solar System.

onmousemove and event object

```
<head>
<meta charset="utf-8">
<title>Mouse events</title>
    <script>
        function UpdateInfo(event) {
            console.log(event); // check the event object properties in console log
            var pageX, pageY;
            // all browsers except IE before version 9
            if ('pageX' in event) {
                pageX = event.pageX;
                pageY = event.pageY;
            }
            else {  // IE before version 9
                pageX = event.clientX + document.documentElement.scrollLeft;
                pageY = event.clientY + document.documentElement.scrollTop;
            }
var message = "screenX: " + event.screenX + ", screenY: " + event.screenY + "<br />";
message += "clientX: " + event.clientX + ", clientY: " + event.clientY + "<br />";
message += "pageX: " + pageX + ", pageY: " + pageY;

            var info = document.getElementById ("info");
            info.innerHTML = message;
        }
    </script>
</head>
<body onmousemove="UpdateInfo(event);">
    <div style="height:1000px;">
      Move your mouse or scroll the document.</div>
    <div id="info" style="height:65px;
      width:200px; left:50px; top:50px;
      background-color:#e0a0a0;
      position:fixed;"></div>
</body>
```

onmousemove.html





JavaScript objects

- JS fundamental and most important datatype is the object
- JavaScript objects are dynamic and can store primitive values or other objects by mapping strings to values
  - Storing ***properties*** like this goes by names as hash map, hashtable, associative array etc.
- Note that objects are ***mutable*** and are manipulated by reference rather than by value
  - If the variable x refers to an object, and the code var y = x; is executed, the variable y holds a reference to the same object, not a copy of that object!

```
//An object is a collection of mostly properties, but methods(function) is also possible
var empty = {};                         // An object with no properties
var o = new Object();                   // Create an empty object: same as {}.
var point = { x:0, y:0 };               // A point object with two properties
var book = { topic: "JavaScript", fat: true };
//Object.create() is a static function that creates new objects
var o1 = Object.create({x:1, y:2});   // o1 inherits properties x and y.
//Access (get or set) the properties of an object with . or []:
book.topic                  // => "JavaScript"
book["fat"]                 // => true: another way to access property values.
book.author = "Flanagan";  // Create new properties by assignment.
book.contents = {};        // {} is an empty object with no properties.
```

Creating and using objects 1

objects.html

```
function objectDemo1(){
      "use strict";
      var book = {
            topic: "Tuxradar PHP",
            year: 2010
      };

      var myBooks = new Array();
      myBooks.push(book); // the push() method adds elements to an array
      // put objects in the array with the static function Object.create()
      myBooks.push(Object.create({topic: "JavaScript", year: 2011}));
      myBooks.push(Object.create({topic: "Android", year: 2012}));

      book.author = "Flanagan"; // Create new properties by assignment
      book.contents = {ch1: "Intro", ch2: "The end"}; // new object in the book object

      // Serializing Objects. Object serialization is the process of converting
      // an object's state to a string from which it can later be restored.
      var js = JSON.stringify(book);  // JSON stands for JavaScript Object Notation
      console.log(js);
      var bookcopy = JSON.parse(js);  // bookcopy is a deep copy (dynamic memory as well) of book
      myBooks.push(bookcopy);         // pushing the JSON bookcopy

      var topics_year = "";
      // Enumerating properties in the associative array with the for/in loop
      for(book in myBooks) {
            topics_year += myBooks[book].topic + ", " + myBooks[book].year + "<br/>";
            // console.log(topics_year);
      }
      document.write("myBooks topics, year:<br />" + topics_year);
```

Creating and using objects 2

```
      // Deleting Properties
      // delete does not operate on the value of the property but on the property itself
      delete bookcopy.topic;           // The book2 object now has no topic property.
      delete bookcopy["year"];         // Now it doesn't have "year" either.
      document.write("<br />bookcopy: " + bookcopy.topic + ", " + bookcopy.year + ", "
           + bookcopy.author + ", " + bookcopy.contents.ch1 + ", " + bookcopy.contents.ch2);
      document.write("<br /><br />JSON bookcopy string: " + js);
      document.write("<br /><br />Date toString: " + (new Date()).toString());
      document.write("<br /><br />Date toJSON: " + (new Date()).toJSON());
      // if there is a built in JSON serialization method
      var jsonDate = (new Date()).toJSON();
      var backToDate = new Date(jsonDate);
      console.log("Serialized date object: " + jsonDate);
      // The valueOf() method returns the primitive value of the specified object.
      var dateVal = (new Date()).valueOf();
      // number of milliseconds in UCT since midnight January 1, 1970
      document.write("<br /><br />Date valueOf: " + dateVal);
      var date = new Date(dateVal);
      document.write("<br /><br />Date valueOf: " + date.toString());
}
myBooks topics, year:
Tuxradar PHP, 2010
JavaScript, 2011
Android, 2012
Tuxradar PHP, 2010
bookcopy: undefined, undefined, Flanagan, Intro, The end
JSON bookcopy string: {"topic":"Tuxradar PHP","year":
      2010,"author":"Flanagan","contents":{"ch1":"Intro","ch2":"The end"}}
Date toString: Sat Feb 25 2012 20:53:33 GMT+0100 (W. Europe Standard Time)
Date toJSON: 2012-02-25T19:53:33.528Z
Date valueOf: 1330199613529
Date valueOf: Sat Feb 25 2012 20:53:33 GMT+0100 (W. Europe Standard Time)
```

objects.html

Arrays, objects and methods

```
//Arrays and objects can hold other arrays and objects:
var points = [              // An array with 2 elements.
 {x:0, y:0},                // Each element is an object.
 {x:1, y:1}
];

var data = {                // An object with 2 properties
trial1: [[1,2], [3,4]],     // The value of each property is an array.
trial2: [[2,3], [4,5]]      // The elements of the arrays are arrays.
};

//When functions are assigned to the properties of an object, we call
//them "methods".  All JavaScript objects have methods:
var a = [];                   // Create an empty array
a.push(1,2,3);                // The push() method adds elements to an array
a.reverse();                  // Another method: reverse the order of elements

//We can define our own methods, too. The "this" keyword refers to the object
//on which the method is defined: in this case, the points array from above.
points.dist = function() { // Define a method to compute distance between points
 var p1 = this[0];       // First element of array we're invoked on
 var p2 = this[1];       // Second element of the "this" object
 var a = p2.x-p1.x;      // Difference in X coordinates
 var b = p2.y-p1.y;      // Difference in Y coordinates
 return Math.sqrt(a*a + // The Pythagorean theorem
                  b*b); // Math.sqrt() computes the square root
};
points.dist();           // => 1.414: distance between our 2 points
```