Content Overview

Event-driven programs and HTML form elements

- Event-driven programs
- `onload`, `onunload`
- HTML forms & attributes
- Button, text box, text area
- Selection list, radio button, check box, password, hidden, ...
- JavaScript form events
- Properties: name, type, value, ...
- Methods: `blur()`, `focus()`, `click()`, ...
- Event handlers: `onblur()`, `onfocus()`, `onchange()`, `onclick()`, ...
- Advanced features & techniques
- Windows, timeouts, cookies
Event-driven Programming

With C++ or Java, programs are usually serially executed. The program starts with main function, executes sequentially from first statement. It may loop or skip sections of code, but the program generally proceeds step-by-step from start to finish.

The programmer specifies the sequence in which execution occurs (with some variability due to input values). There is a beginning and an end to program execution.
Computation within a Web page is rarely serial, instead the page reacts to events such as mouse clicks, buttons, ... Much of JavaScript’s utility is in specifying actions that are to occur in the page as a result of some event.

The programmer may have little or no control over when code will (if ever) be executed, e.g. code that reacts to a button click. There is no set sequence, the webpage waits for events and reacts.
onload and onunload

The simplest events that a web browser can react to occur when a webpage is loaded or unloaded.

A web programmer can use the `onload` attribute of the `<body>` tag, which specifies JavaScript code to be executed when the page is first loaded.

The `onunload` attribute similarly specifies JavaScript code that is to be automatically executed when the browser leaves the page. (This event handler is rarely used.)

A simple example follows.

Note: Not every browser seems to pay attention to the `onunload` attribute.
onload and onunload Example

```
<html>
<!-- COMP519 form01.html 2015.10.05 -->
<head>
    <title>Hello/Goodbye page</title>
    <script>
        function Hello()
        {
            globalName=prompt("Welcome to my page. " +
                                "What is your name?",""");
        }
        
        function Goodbye()
        {
            alert("So long, " + globalName +
                        " come back real soon.");
        }
    </script>
</head>
<body onload="Hello();" onunload="Goodbye();">
    <p>Whatever text appears in the page.
        ........
    </p>
</body>
</html>
```
The attributes used on the previous example are the “old” style of assigning event handlers.

The “modern” way to assign event handlers is to use a JavaScript function to add a handler or a “listener”.

The way to do this in JavaScript is shown on the next page.
Event Handlers Using a JavaScript Method

```html
<html>
<!-- COMP519  form01eh.html  2015.10.05 -->
<head>
    <title>Hello/Goodbye page</title>
    <script>
        function Hello()
        {
            globalName=prompt("Welcome to my page. " + "What is your name?","" );
        }

        function Goodbye()
        {
            alert("So long, " + globalName + " come back real soon.");
        }
    </script>
</head>
<body>
    <p>Whatever text appears in the page.
        ........
    </p>
    
    <script>
        window.addEventListener("load", Hello, false);
        window.addEventListener("unload", Goodbye, false);
    </script>
</body>
</html>
```
Actually things are more complicated. The method mentioned doesn’t work on all browsers. Internet Explorer uses a different JavaScript method to add an event handler.

A more robust method (that requires writing more code) is to use the following `<script>` block in place of the previous one.

```html
<script>
    if (window.addEventListener) // ** Non-IE browsers
    { window.addEventListener("load", Hello, false); }
    else // ** IE browsers
    { window.attachEvent("onload", Hello); }
</script>
```

This is a situation where the use of jQuery can help with the cross-browser compatibility issue. (But I will leave those details for others to look up if they wish.)
Of course, you would also write similar code for the `onunload` event handler.

In the examples that follow, I will be using a mix of the “old” and the “modern” method to assign event handlers.

While I won’t show the more complicated (cross-browser) method for assigning the event handler (when using the “modern” method) on the code examples, I will be using them in the actual code that I write on the webpages that I demonstrate.
HTML Forms

Most event-handling in JavaScript is associated with form elements. An HTML form is a collection of elements for handling input, output, and events in a webpage. You place a `<form>` element in the `<body>` element of the webpage.

```
<form id="FormName">
  ...
  ...
</form>
```

Form elements might include:
for input: button, selection list, radio button, check box, password, ...
for input/output: text box, text area, ...

We will also encounter forms when we consider CGI programming and PHP. A form groups together elements, whose contents can be submitted together to a server for further processing on the server.
**HTML Forms (cont.)**

`document.forms[]` is a JavaScript (associative) array that will contain elements for each form on the webpage. This array uses the "id" associated with the form as the array index (assuming that a "id" is defined).

One can also use the syntax `document.forms[0]` for the first form on the page (and higher indices if there is more than one form on the same webpage).

Using the "dotted" syntax, we can then access other HTML elements of the form (using their ids given to those elements). HTML pages are stored as a hierarchy of parent/child relationships, and this defines the way to access the elements.

(More examples to come…)
Strictly speaking, most of the examples shown here do not require the use of the `<form>` element, but I will be using that element in any event. Forms are typically used when information is submitted to a server, and the examples we show won't involve submitting information in this fashion.

I will still use the forms for logically grouping HTML elements together.
Button Element

The simplest form element is a button.

This is analogous to a real-world button, in that clicking the button can be used to trigger events to happen.

Typical syntax is as follows:

```html
<input type="button" value="LABEL" onclick="JAVASCRIPT CODE">
```

Or using the modern event listener syntax, you define the button:

```html
<input type="button" value="LABEL" id="myButton">
```

and then assign the event listener to the button:

```html
<script>
    var b = document.getElementById("myButton");
    b.addEventListener("click", myFunction);
</script>
```

where "myFunction" is a JavaScript function to perform some action(s).
<html>
<!-- COMP519 form02eh.html 2015.10.12 -->
<head>
  <title>Fun with Buttons</title>
  <script src="http://cgi.csc.liv.ac.uk/~martin/teaching/comp519/JS/random.js"></script>
  <script>
    function popup()
    {
      var num = randomInt(1,100);
      alert('The lucky number for the day is ' + num);
    }
  </script>
</head>
<body>
  <form id="ButtonForm">
    <p>
      <input type="button" value="Click for Lucky Number" id="Random">
    </p>
  </form>
  <script>
    var el = document.getElementById("Random");
    if (el.addEventListener)
    {
      el.addEventListener("click", popup, false);
    }
    else
    {
      el.attachEvent("onclick", popup);
    }
  </script>
</body>
</html>
Buttons and Writing to Windows

Alert boxes are fine for displaying short, infrequent messages. Not well-suited for displaying longer, formatted text. Not integrated into the page, requires the user to explicitly close the alert box.

QUESTION: could we instead use `document.write` command in the previous example?

NO – We would overwrite the entire current page.

But we could open a new browser window and write there. Some commands used to deal with other windows.

```
var OutputWindow = window.open(); // open a window and
// assign a name to that object
// (first arg is an HREF)
OutputWindow.document.open(); // open that window for writing
OutputWindow.document.write("WHATEVER"); // write text to that
// window
OutputWindow.document.close(); // close the window
```
A Window Example

<html>
<!-- COMP519 form04eh.html 2015.10.12 -->
<head>  
<title> Fun with Buttons </title>  
<script>
    function Help()
        // Results: displays a help message in a separate window
        { var OutputWindow = window.open();
            OutputWindow.document.open();
            OutputWindow.document.write("This might be a context-sensitive help message, depending on the " + 
                "application and state of the page.");
        }
    </script>  
</head>  
<body>
    <form id="ButtonForm">
        <p>  
            <input type="button" value="Click for Help" id="myButton">  
        </p>
    </form>
    <script>  
        // ** JavaScript for event handler
    </script>
</body>  
</html>
You can provide arguments to `window.open` to specify properties to the new window.

The first argument can specify a hyperreference.

The second argument can specify an internal name to be used in the JavaScript.

The third argument can specify window properties like the size of the window.

The example on the next page demonstrates this.
<html>
<!-- COMP519  form05eh.html  2015.10.12 -->
<head> <title> Fun with Buttons </title> <script>
      function Help()
      // Results: displays a help message in a separate window
      { var OutputWindow =
        window.open("","","status=0,menubar=0,height=200,width=200");
        OutputWindow.document.open();
        OutputWindow.document.write("This might be a context-sensitive help message, depending on the " +
        "application and state of the page.");
      }
    </script> </head>
<body>
<form id="ButtonForm">
  <p> <input type="button" value="Click for Help" id="myButton"> </p>
</form>
<script> //**  JavaScript for event handler
</script>
</body>
</html>
A text box allows for user input (and could also be used for output).

Unlike prompt, user input persists on the page and can be edited by the user.

```html
<input type="text" id="Box" name="Box" ... />
```

Optional attributes:
  - size : width of the box (number of characters)
  - value : initial contents of the box

JavaScript code can access the contents in the example that follows as `document.forms[‘BoxForm’].userName.value`

(Using the id, we can access the input in a different fashion.)
Text Box Example

<html>
<!-- COMP519 form06eh.html 2015.10.13 -->
<head> <title> Fun with Text Boxes </title>
    <script>
        function greetings() {
            alert('Thanks, ' +
                document.forms['BoxForm'].userName.value + ', I needed that.');
        }
    </script>
</head>
<body>
    <form id="BoxForm">
        <p> <label for="userName">Enter your name here:</label>  
            <input type="text" id="userName" size="12" value="">  
        </p>
        <input type="button" value="Click Me" id="myButton">  
    </form>
    <script> //** Code to attach event listener  
    </script>
</body>
</html>
As well as reading from a text box, we can also change the contents of a text box with an assignment statement.

Note: the contents of a text box is raw text, no HTML formatting is performed.

Also: contents are accessed as a string, so we must `parseFloat` or `parseInt` if we want to treat input as a number.
Other Text Box Events

There are other events that trigger upon certain actions, and can trigger and be captured by event handlers.

The **change** (or **onchange** for IE) event triggers when the contents of a text box are changed.

The **focus** (or **onfocus** for IE) event triggers when the mouse clicks in the text box or the keyboard is used to select the text box (such as using the "tab" key to do so).

The **blur()** function can be used to remove the focus from a text box.

The next example uses the "old style" event handlers.
function FahrToCelsius(tempInFahr)
    // Assumes: tempInFahr is a number (degrees Fahrenheit)
    // Returns: corresponding temperature in degrees Celsius
    { return (5/9)*(tempInFahr - 32); }
Text Box Validation

What if a user enters non-numeric data in the Fahrenheit box?

Solution: Writing a function that will perform some validation. Start with legal value (or an empty text box).
At change (or onchange for IE), verify that new value is legal (otherwise, reset the input box).

The verify.js library defines several functions for validating text boxes.

(http://www.csc.liv.ac.uk/~martin/teaching/comp519/JS/verify.js).

```javascript
function VerifyNum(textBox)
// Assumes: textBox is a text box
// Returns: true if textBox contains a number, else false + alert
{
    var boxValue = parseFloat(textBox.value);
    if ( isNaN(boxValue) ) { // ** isNaN function
        alert("You must enter a number value!");
        textBox.value = "";
        return false;
    }
    return true;
}
```
<html>
<!-- COMP519 form09.html 2015.10.13 -->
<head>
<title>Fun with Text Boxes</title>
<script src="http://www.csc.liv.ac.uk/~martin/teaching/comp519/JS/verify.js"></script>
<script>
function FahrToCelsius(tempInFahr)
{
    return (5/9) * (tempInFahr - 32);
}
</script>
</head>
<body>
<form id="BoxForm">
<p><label for="Fahr">Temperature in Fahrenheit:</label>
<input type="text" id="Fahr" size="10" value="0"
onchange="if (verifyNum(this)) { // "this" refers to current element
    document.forms["BoxForm"].Celsius.value =
    FahrToCelsius(parseFloat(this.value)); }">

&nbsp;&nbsp;----&gt;&nbsp;&nbsp;</p>
<input type="text" id="Celsius"
    size="10" value="" onfocus="blur();"/>
in Celsius
</form>
</body>
</html>

(view page) (This example could be altered to use “new”-style event handlers.)
Text Areas

A text box is limited to one line of input/output.

A "text area" is similar to a text box in functionality, but can specify any number of rows and columns.

<textarea id="TextAreaName" rows="NumRows" cols="NumCols"> Initial Text </textarea>

Note: unlike a text box, a text area has a separate closing tag.

The initial contents of the text area appear between the two text area tags.

As with a text box, no HTML formatting of text area contents is performed.
Text Box Example

```html
<html> <!-- COMP519 form10.html 2015.10.15 -->
<head>  
<title> Fun with Textareas </title>
<script src="http://www.csc.liv.ac.uk/~martin/teaching/comp519/JS/verify.js"></script>
</head>
<body>  
<form id="AreaForm">
<text>
Show the numbers from <input type="text" id="lowRange" size = "4" value="1">

to <input type="text" id="highRange" size="4" value="10">

raised to the power of <input type="text" id="power" size="3" value="2">

</text>
</body>
</html>
```
<input type="button" value="Generate Table" id="myButton">
<br> <br>
<textarea id="Output" rows="20" cols="15"></textarea></p>
</div>
</form>
<script>
var lowIn = document.getElementById("lowRange");
var highIn = document.getElementById("highRange");
var powerIn = document.getElementById("power");
var myButton = document.getElementById("myButton");

if (lowIn.addEventListener)
{   lowIn.addEventListener("change",
    function (){ verifyInt(lowIn); }, false); }
else
{   lowIn.attachEvent("onchange",
    function (){ verifyInt(lowIn); }); }
...

... // Add other event handlers
</script>
</body>   </html>
The previous examples have illustrated both ways to access HTML elements on a webpage.

Mostly, we have been accessing data input fields by giving them ids, and using the “dotted” syntax from the Document Object Model tree structure.

What if someone modifies the HTML document?

Then, all those multiply-referenced items can no longer be accessed if the name of the form changes.
A more reliable manner (more resistant to changes in the webpage code) would be to give each element an ID (using the "id" attribute) and use the JavaScript `document.getElementById` method. (This method was used in the JavaScript code to attach events using the JavaScript functions to do so.)

In practice, every item (like input boxes, textboxes, etc) should be given a unique id.

The “name” attribute (which I haven’t been supplying in the examples so far) is used if/when information is submitted to the server, and the “id” is what is used on the client to identify that HTML element. (Typically, the name and id are given the same value.)
Check boxes

A list of check boxes, is a set of items, one or more of which can be selected.

This is easy to create in HTML, simply using the HTML element:

```html
<input type="checkbox">
```

Give the checkbox a `name` and use this for each of the elements in the checkbox list, with a separate `value` for each checkbox item.

The set of checkboxes are stored internally as an array.
Checkbox example

```html
<html>
<!-- COMP519 form16eh.html 2015.10.14 -->
<head>
  <title> Check Boxes </title>
  <script>
    function processCB() {
      var boxes = document.forms['BoxForm'].cb.length;
      var s="";
      for (var i = 0; i < boxes; i++) {
        if (document.forms['BoxForm'].cb[i].checked) {
          s = s + document.forms['BoxForm'].cb[i].value + " ";
        }
      }
      if (s == "") { s = "nothing"; }
      alert("You selected " + s);
    }
  </script>
</head>
```
Which of these things is unavoidable in life (select one or more)?

- Death
- Taxes
- Robbie Williams

<input type="button" value="Done" id="myButton">

<script>
  var b = document.getElementById("myButton");
  if (b.addEventListener) {
    b.addEventListener("click", processCB, false);
  } else {
    b.attachEvent("onclick", processCB);
  }
</script>
Radio Buttons

Radio buttons are similar to check boxes, but only one of them can be selected at any time.

They are defined by `<input type="radio">` tags (similar to the checkbox tags in the previous example, with similar properties) and accessed in the same manner.
Form Submission

Typically forms are used when information is submitted to a server.

This involves either having a “submit” button in the form, or using JavaScript to submit the form by calling the submit function.

When you submit the form, you must specify an “action” to be performed (this is usually a script of some type, such as a PHP script) and a method of sending the information (either “get” or “post”).

We will see more of this later...
Form Submission Example

```
<html>
<!-- COMP519 form17.html 2015.10.15 -->
<head>
<title> Submitting information </title>
</head>
<body>
<form id="Information" action="process_form.php" method="post">
  
  <p><br>  
  <label for="Person">Please enter your name:</label>  
  <input type="text" size="30" name="Person" id="Person">
  </p>
  
  Please select your gender:
  
  <input type="radio" name="sex" id="female" value="female">  
  <label for="female">Female</label>  
  <br>
  <input type="radio" name="sex" id="male" value="male">  
  <label for="male">Male</label>  
  <br>
  <input type="submit" value="Submit">

</form>
</body> 
</html>
```
The PHP Backend for the Previous Example

```html
<!DOCTYPE HTML>
<html>
<head>
<title>PHP very simple form example</title>
</head>

<body>
<?php
$name = $_POST["Person"];  
$gender = $_POST["sex"];  

echo "<p>Hello " . $name . "!";</p>
if($gender =="male")
    { echo "<p>What’s a fine fellow doing on a website like this one?\n</p>\n";  } else
    { echo "<p>Welcome! What’s a fine women like you doing on a website like this one?\n</p>\n";  }
?>
</body>
</html>
```
JavaScript Timeouts

The `setTimeout` function can be used to execute code at a later time.

```
setTimeout(JavaScriptCodeToBeExecuted, MillisecondsUntilExecution)
```

Example: forward a link to a moved page.

```html
<html> <!-- COMP519 form13eh.html 2015.10.15 -->
<head>  
  <title> Fun with Timeouts </title>
  <script>
    function Move()
    // Results: sets the current page contents to be newhome.html
    { self.location.href = "newhome.html"; }
  </script>  
<body>
  <p>This page has moved to <a href="newhome.html">newhome.html</a>.
  </p>
  <script>
    if (window.addEventListener)
    { window.addEventListener("load", Move, false); }
    else
    { window.attachEvent("onload", Move); }
  </script>
</body>  
</html>
```
Another Timeout Example

```html
<html>
<!-- COMP519 form14eh.html 2015.10.15 -->
<head>  
<title> Fun with Timeouts </title>
<script>
    function timeUntil() {
        // Assumes: input area with id "countdown" exists in the page
        // Results: every second, recursively writes current countdown in
        {
            // CODE FOR DETERMINING NUMBER OF DAYS, HOURS, MINUTES, AND SECONDS
            // UNTIL GRADUATION (see the file for this code!!)
            var display = document.getElementById("countdown");
            display.value = days + " days, " + hours + " hours, " + minutes + " minutes, and " + secs + " seconds";
            setTimeout("timeUntil();", 1000);
        }
    </script>  </head>
<body>
<form id="CountForm">
    <div style="text-align: center;">
        <p> Countdown to Graduation 2016<br>
            <textarea id="countdown" id="countdown" rows="5" cols="15"
                style="font-family: Courier;" onfocus="blur();"></textarea>
        </p>
    </div>
</form>  
<script> /* Event handlers */ </script>
</body>  </html>
```
Cookies and JavaScript

Cookies are small data files stored on the client machine. They can be accessed and/or modified by server requests as webpages are loaded. They can also be accessed and/or modified directly by JavaScript code in a webpage.

Potential applications:
e-commerce: remember customer name, past visits/purchases, password, . . .
tutorials: remember past experience, performance on quizzes, . . .
games: remember high score, best times, . . .
Each Web page can have its own cookie. 
**document.cookie** is a string of **attribute=value** pairs, separated by semi-colons.

"**userName=Dave;score=100;expires=Mon, 21-Feb-14 00:00:01 GMT**"

As mentioned, JavaScript has methods to set, modify, and delete cookies.
function getCookie(Attribute)
// Assumes: Attribute is a string
// Results: gets the value stored under the Attribute
{   if (document.cookie.indexOf(Attribute+"=") == -1)
    {   return "";   }
else {
    var begin = document.cookie.indexOf(Attribute+"=") + Attribute.length+1;
    var end = document.cookie.indexOf(";", begin);
    if (end == -1) end = document.cookie.length;
    return unescape(document.cookie.substring(begin, end)); }
}

function setCookie(Attribute, Value)
// Assumes: Attribute is a string
// Results: stores Value under the name Attribute, expires in 30 days
{   var ExpireDate = new Date();
    ExpireDate.setTime(ExpireDate.getTime() + (30*24*3600*1000));
    document.cookie = Attribute + "=" + escape(Value) + "; expires=" + ExpireDate.toGMTString();
}

function delCookie(Attribute)
// Assumes: Attribute is a string
// Results: removes the cookie value under the name Attribute
{   var now = new Date();
    document.cookie = Attribute + "=; expires=" + now.toGMTString();
}
Cookie Example

```html
<html>
<!-- COMP519  form15.html  2015.10.16 -->

<head>
  <title> Fun with Cookies </title>
  <script
src="http://www.csc.liv.ac.uk/~martin/teaching/comp519/JS/cookie.js">
  </script>
  <script>
    function Greeting()
    // Results: displays greeting using cookie
    {
      visitCount = getCookie("visits");
      if (visitCount == ") {
        alert("Welcome to my page, newbie.");
        setCookie("visits", 1);
      }
    else {
      visitCount = parseFloat(visitCount)+1;
      alert("Welcome back for visit #" + visitCount);
      setCookie("visits", visitCount);
    }
  }
</script>
</head>
```
<body>
<p>Here is the stuff in my page. </p>
<form id="ClearForm">
<div style="text-align: center;">
<p> <input type="button" value="Clear Cookie" id="clear">
</p>
</div>
</form>
<script>
if (window.addEventListener)
{ window.addEventListener("load", Greeting, false); } else
{ window.attachEvent("onload", Greeting); }
var b = document.getElementById("Clear");
if (b.addEventListener)
{ b.addEventListener("click", 
    function () { delCookie('visits'); }, false); }
else
{ b.attachEvent("onclick", 
    function () { delCookie('visits'); }); }
</script>
</body>
</html>
A Recap of Things We Have Seen

Event-driven programs

- event handlers for when page loads/unloads, buttons are clicked, input boxes change, etc.

HTML forms & attributes

- button, text box, text area
- radio buttons, check boxes, selection list (use `<select>` tag)

As with most topics in this module, there is far too much for me to discuss everything in the lectures.

The most fundamental things to understand are the ideas behind event-driven programming, what events can be captured, and how to perform that task.