JavaScript Events and Forms

This workbook provides a basic introduction to JavaScript Events and using them with HTML forms. It assumes that you have already completed the Introduction to JavaScript and the JavaScript and the Document Object Model worksheets. This workbook follows the conventions used in the previous workbooks, it includes explanations, code descriptions and code examples, which you should read. It also includes activity sections which ask you to carry out a set of actions, for example to create or modify some code. The activities reinforce what you have read, but also give you vital coding practice. The workbook is design to be read in sequence, if you skip parts it may make it harder to understand later parts.

Whilst the workbook includes everything you need to cover, you may find it useful to refer to other sources of information. The w3Schools website has a course on JavaScript, as does the codeacademy website. Our library also has books on JavaScript.

JavaScript Events

The previous workbooks have introduced JavaScript and looked at how we can manipulate the DOM tree using JavaScript, but the examples weren't really interactive. There wasn't anything for the user to do, the code just ran. In this workbook we will look at adding user interactivity. JavaScript can identify a number of 'events' caused by user interaction. These events can be used to trigger functions. These functions carry out actions and provide a response to the user.

There are a number of different events that we can use to trigger our JavaScript functions. They fall into six main types:

- User Interface Events – when the user interacts with the browser interface, e.g. they resize the browser.
- Keyboard Events – when the user interacts with the keyboard, e.g. when the user presses a key.
- Mouse Events – when the user interacts with the mouse, trackpad or touchscreen e.g. when the user clicks on an element.
- Focus Events – when an element gains of loses focus, e.g. an entry field on a form.
- Form Events – when the user interacts with a form element, e.g. when the user selects a checkbox.
- Mutation Events – when the DOM Tree has been modified, e.g. when an Element Node has been removed.

In order to attach an event to our HTML page we need to select the relevant Element Node, bind the particular events we want to use to the Element Node, and specify what code is run when the event is triggered.

There are three ways to bind an event to an element, HTML Event Handlers, DOM Event Handlers and DOM Event Listeners. HTML Event Handlers are now considered bad practice and DOM Event Handlers rather dated, so we will only look at DOM Event Listeners (though these may not be supported by older browsers).

We can add a listener to an Element Node using its `addEventListener()` method,

```
element.addEventListener('event', functionName, Boolean);
```

`element` is the Element Node we wish to attach the listener to.

`'event'` is the particular event we want to listen for.

`functionName` is the name of the function we wish to call if the event is triggered

`Boolean` indicates the event flow. We will say more about this later. It is usually set to false

Let's consider a brief example to see how this works. Create and upload the following HTML and CSS files to the server.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Catalogue</title>

    <link rel="stylesheet" href="domcss.css">
    <script src="events.js"></script>
  </head>

  <body>
    <div id="page">
      <h1 id="header">Course Catalogue</h1>
      <h2>Course Status</h2>
      <ul>
        <li id="one" class="running"><em>new</em> BSc Computer
Networking</li>
        <li id="two" class="running">BSc Computing</li>
        <li id="three" class="running">BSc Informatics</li>
        <li id="four">BSc Mechatronics</li>
      </ul>
      <script>doEvents();</script>
    </div>
  </body>
</html>
```

```
body {
background-color: hsl(0, 0%, 0%);
font-family: sans-serif;
margin: 0;
padding: 0;}

#page {
background-color: hsl(290, 5%, 50%);
margin: 0 auto 0 auto;}

h1 {
color: hsl(197, 50%, 90%);
margin: 0 auto 0 auto;
padding: 30px 10px 20px 10px;
```

```
    text-shadow: 2px 2px 1px hsl(290, 5%, 35%);}

    h2 {
    color: hsl(176, 50%, 90%);
    margin: 0 0 10px 0;
    padding: 0px 10px 20px 10px;
    text-shadow: 2px 2px 1px hsl(290, 5%, 35%);}

    ul {
    background-color: hsl(290, 5%, 50%);
    border: none;
    padding: 0;
    margin: 0;}

    li {
    background-color: hsl(290, 5%, 80%);
    color: hsl(360, 100%, 100%);
    border-top: 1px solid hsl(290, 5%, 90%);
    border-bottom: 1px solid hsl(290, 5%, 70%);
    list-style-type: none;
    text-shadow: 2px 2px 1px hsl(290, 5%, 70%);
    padding-left: 1em;
    padding-top: 10px;
    padding-bottom: 10px;}

    .running {
    background-color: hsl(155, 60%, 50%);
    text-shadow: 2px 2px 1px hsl(155, 60%, 40%);
    border-top: 1px solid hsl(155, 60%, 60%);
    border-bottom: 1px solid hsl(155, 60%, 40%);}

    .full {
    background-color: hsl(12, 60%, 50%);
    text-shadow: 2px 2px 1px hsl(12, 60%, 40%);
    border-top: 1px solid hsl(12, 60%, 60%);
    border-bottom: 1px solid hsl(12, 60%, 40%);}
```

We are now going to attach an Event Listener to the Node Element with id='two'

```
function doEvents(){
  var el = document.getElementById('two');

  el.addEventListener('click', toggleStatus, false);
}
```

So, the function `toggleStatus()` will run when the user (single) clicks on the specified Element Node. We also need to define `toggleStatus()`

```
function toggleStatus(){
  if (this.className === 'full'){
    this.className = 'running';}
  else if (this.className === 'running'){
    this.className = 'full';
  }
}
```

Note the use of the keyword `this` to refer to the Element Node to which the Event Listener is attached.

Activity

1. Create a new JavaScript file containing the two functions above.
2. Click on the BSc Computing course and observe the effect.
3. Modify `doEvents()` so that it applies the Event Listener to all the list items (use a loop).
4. Modify `toggleStatus()` so that a mouse click changes a non-running course to a running course; a running course to a full course; and a full course to a non-running course.

We mentioned Event Flow earlier, noting that it is a Boolean value and is part of the `addEventListener()` method. In the example below, it is set to `false`.

```
el.addEventListener('click', toggleStatus, false);
```

Event Flow is only a concern when we have Event Listeners on Element Nodes which are in an ancestor/decedent relationship with each other in the DOM Tree). For example in the HTML above, suppose we added an Event Listener on the `<ul>` as well, which also fired on click. Because the `<li>` is nested in the `<ul>` any click on the `<li>` is also a click on the `<ul>`, so two Event Listeners will fire. If there was a similar Event Listener on the `<body>`, then three Event Listeners would fire. Event Flow determines the order in which they should fire. Event Bubbling (Event Flow = `false`) fires the Event Listener on the most deeply nested Element Node first (`<li>`), then the second most deeply nested (`<ul>`), then the next most (`<body>`). Event Capturing (Event Flow = `true`) fires the Event Listener on the least nested Element Node first (`<body>`), then the next least (`<ul>`), then then next least (`<li>`). Working out the logic for nested Event Listeners can be complex, so they should be avoided except where they are actually useful. Generally, we can just ignore it and set it to `false`.

Usually we won't be attaching Event Listeners to list items or other bits of text content, we will be attaching them to elements that have some clear interactive function, like buttons and form items. We will look at this next.

Create and upload the following HTML file to the server.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Form</title>
    <link rel="stylesheet" href="eventcss.css" />
  </head>
  <body>
    <div id="page">
      <h1>Course Catalogue</h1>
      <h2>Register Account</h2>
      <form action="FormEcho.php" method="post" id="account">

        <label for="username">Create a username: </label>
        <input type="text" id="username" name="username"><div
id="feedback"></div>

        <label for="password">Create a password: </label>
        <input type="password" id="password" name="password">

        <input type="submit" id = 'submit' value="Create">

      </form>
    </div>
    <script src="events.js"></script>
  </body>
</html>
```

Hopefully this looks fairly familiar. It displays a text input box with the label "Create a username" and a password input box with the label "Create a password". There is a submit button labelled "Create" and the information from the form is submitted to FormEcho.php using the "post" method. FormEcho.php just writes the values submitted to the screen (you may remember we used this when we learned about HTML forms in the first year).

Note the `div` with `id="feedback"` which currently has no content.

Create and upload the following CSS to the server. Link it to your HTML page.

```
body {
      background-color: hsl(0, 0%, 0%);
      font-family: sans-serif;
      margin: 0;
      padding: 0;}

#page {
      background-color: hsl(290, 5%, 50%);
      margin: 0 auto 0 auto;}

h1 {
      color: hsl(197, 50%, 90%);
      margin: 0 auto 0 auto;
      padding: 30px 10px 20px 10px;
      text-shadow: 2px 2px 1px hsl(290, 5%, 35%);}

h2 {
      color: hsl(176, 50%, 90%);
      margin: 0 0 10px 0;
      padding: 0px 10px 20px 10px;
      text-shadow: 2px 2px 1px hsl(290, 5%, 35%);}

form {
      padding: 0 60px 65px 60px;}

label {
      color: hsl(360, 90%, 90%);
      display: block;
      margin: 10px 0 10px 0;
      font-size: 20px;}
```

```
input[type='text'], input[type='password'] {
      background-color: hsl(290, 5%, 70%);
      color: hsl(0, 0%, 30%);
      font-size: 20px;
      width: 96%;
      padding: 4px 6px;
      border: 1px solid hsl(290, 5%, 70%);}

input[type='text']:focus, input[type='password']:focus {
      border: 1px solid hsl(0, 0%, 90%);
      background-color: hsl(0, 0%, 90%);
      outline: none;}

input[type='submit'] {
      background-color: hsl(120, 35%, 50%);
      color: hsl(100, 60%, 85%);
      border: none;
      border-radius: 5px;
      padding: 8px 10px;
      margin-top: 10px;
      float: right;
      font-size: 18px;
      text-decoration: none;}

input[type='submit']:hover {
      background-color: hsl(120, 55%, 50%);
      color: hsl(100, 60%, 85%);
      cursor: pointer;
      box-shadow: none;
      position: relative;}

#feedback {
      color: hsl(360, 70%, 70%);
      padding: 10px 0 0 22px;}
```

It is worth taking a look at this CSS and in particular how we are styling form elements using different types of CSS selector.

`label` – styling the label displayed on the screen with the input widget.

`input[type='text']` - styling the text type input widget

`input[type='text']:focus` – styling the text type input widget when it is the input focus

`input[type='submit']` – styling the submit button

`input[type='submit']:hover` – styling the submit button when the cursor hovers over the button

Now that we have our form and it is looking reasonable attractive, we can enhance it with some Event Listeners.

Let's suppose that we want the username to be at least five characters long. We can attach an Event Listener to the username text widget, so that when focus us lost (i.e. the user clicks in the password widget) the length of the username is checked and, if necessary, an error message is displayed.

Create and upload the following JavaScript file to the server. Make sure the JS file is called events.js

```
var elUsername = document.getElementById('username');
var elFeedback = document.getElementById('feedback');

function checkUsername(minLength) {
  if (elUsername.value.length < minLength) {
    elFeedback.textContent = 'Username must be ' + minLength + ' characters
or more';
  } else {
    elMsg.textContent = '';
  }
}

elUsername.addEventListener('blur', function() {
  checkUsername(5);
}, false);
```

Accessing the Element Nodes and the `checkUserName()` function should make sense to you, but it is worth taking a closer look at the function call in the Event Listener.

```
elUsername.addEventListener('blur', function() {
  checkUsername(5);
}, false);
```

We might have imagined coding this as:

```
elUsername.addEventListener('blur', checkUsername(5), false);
```

However, if we did this the `checkUsername(5)` would execute as soon as the Event Listener was added. We only want it to be executed when the event triggers it. We therefore need to use an anonymous function (indicated by the `function()` keyword) which isn't executed immediately. We then make this anonymous function call the function we want to use, in this case `checkUsername()`, and we are passing 5 as a parameter.

It is also worth noticing how we are accessing the text that has been typed into the username text field. We can select the Element Node using its id

```
var elUsername = document.getElementById('username');
```

and then use the `.value` property to get the string that was entered by the user

```
elUsername.value
```

and then use the string `.length` property to gets its length

```
elUsername.value.length
```

You can also now see the purpose of the empty `<div>`.

Activity

5. Check to see what the HTML form does under the following circumstances, reload the form between each test.
6. Enter a username a five or more characters long, click in the password box.
7. Enter a username less than five characters, click in the password box.
8. Click in the username text box, click in the password text box.
9. Type in a password and click the Create button.
10. Enter a username less than five characters, click the Create button.

There are a couple of things to note from these tests. Firstly, if the username text box has never been in focus, it cannot lose focus (the blur event does not occur). Secondly, clicking on the Create button is still possible, even if the username is too short, so we would need to add further functionality to stop invalid data being posted to whatever database we are using.

Activity

11. Add an Event Listener to the password entry field, that checks the password is at least 8 characters long. You should use the existing `checkUsername()` function, but you will need to add a div to the HTML and make sure the error message is generic.
12. Check to make sure the two entry fields behave correctly.

We are now going to add a checkbox to show/hide the password and we will also take a quick look at how we handle errors.

Activity

13. Add a checkbox to your HTML

```
<input type="checkbox" id="seePassword" name="seePassword">
<label for="seePassword">See Password </label>
```

14. Add the following Event Listener to the checkbox

```
chkSeePw.addEventListener('change', function(e) {
  var target = e.target;
  try {
    if (target.checked) {
      elPassword.type = 'text';
    }
    else {
      elPassword.type = 'password';
    }
  } catch(error) {
      alert('This browser cannot switch type');
    }
}, false);
```

15. Check to make sure that the checkbox behaves correctly and that the length checking on username and password still behave correctly.

There are several new things we need to take a look at here.

Firstly there is a new trigger `'change'`, which is a Form Event which fires when a select box, checkbox or radio button changes.

Secondly, we have included the code we wish to trigger within the `.addEventListener()` parameters, rather than calling a separate function.

Thirdly, our anonymous function now appears to be passing a parameter, even though we said it couldn't. This is because when an Event Listener calls its triggered function, it automatically passes a reference to the Event Object to which it was attached. This allows us to use the properties and methods associated with the Event Object in the function. We need to associate a variable name with the reference, and `e` (for event) is typically chosen. How we do this depends on whether or not we need to pass other parameters as well as the Event Object.

If we aren't passing any other parameters, we can do this:

```
function someFunction(e) {
var target = e.target; }

el.addEventListener('blur', someFunction, false);
```

If we need to pass additional parameters, we can do this:

```
function someFunction(e, minLength) {
var target = e.target; }

el.addEventListener('blur', function(e) {someFunction(e, 5);},
false);
```

Remember, we cannot include other parameters in the anonymous function otherwise it will fire as soon as the Event Listener is added.

Fourthly, we are accessing and changing a form control property, `.type` to change the password field from `type = 'password'` to `type = 'text'` in order to reveal and hide the characters in the password. Form controls have a number of useful properties and methods associated with them.

Finally we have the error handling. In this case we are trapping an error which could be caused if the user is using an old browser which doesn't support the changing of a form controls `.type` property. We can use error handling to trap all sorts of potential errors in our code. The error handling follows this model:

```
try {
  // try to execute this code
} catch (error) {
  // if there is an error, run this code
} finally {
  // always run this code
}
```

In the password example above, if the `elPassword.type` cannot be set, then an alert is displayed to the user, `alert('This browser cannot switch type');` This will create a new pop-up alert box containing the message string.

We can also throw our own errors, which can then be trapped.

Activity

16. Add the following code to the `chkSeePw` Event Listener code immediately after `elPassword.type = 'text';`

    ```
    throw new Error('People can see your password!');
    ```

17. Check to see what happens when you check the checkbox.

The reason we get the `'This browser cannot switch type'` message is because the `Error` we have thrown is caught by the `catch` statement which in turn causes the `alert`. The pop-up alert box is being created by the `alert`, not by the `throw new Error`. If we want to display the message associated with the `Error`, we can change the `alert` string to `alert(error.message);`

JavaScript has several different types of Error Object which can be thrown and caught. `Error` is the most generic.

At the moment our form will allow the user to click the create button, even if the form has errors, or they have already submitted it. We will next look at preventing the first case, submission when the form has errors.

Activity

18. Amend the username and password Event Listeners to fire on `'input'` instead of `'blur'`
19. Check to see what difference this has made.
20. Add the following to the JavaScript file

    ```
    submit.disabled = true;

    submit.className = 'disabled';
    ```

    This disables the submit button and changes its class to `.disabled` so we can style it differently.

21. Add some CSS rules to style `#submit.disabled` so we can tell when the button is disabled
22. Add some CSS rules to `#submit.disabled:hover` so we can tell when the button is disabled, include `cursor: not-allowed;`
23. Modify the password/username checking function so that the submit button will only be enabled

    ```
    submit.disabled = false;

    submit.className = '';
    ```

    when the password is at least 8 characters AND the username is at least five characters AND the form has not already been submitted (something like `(unOK && pwOK && !submitted)` using three variables to track whether the conditions are met).

24. Check that your code works by entering different length usernames/passwords, deleting characters to make fields invalid again, and so on. The submit button should only be enabled

when all three conditions are met. It should be disabled again if the username or password is made invalid by subsequent deletions.

Now we will next look at preventing the second case, submission when the form has already been submitted.

Activity

25. Add a new Event Listener to the form (`id = "account"`) that includes the following code (in an anonymous function, pass e):

```
if (submit.disabled || submitted){
  e.preventDefault(); // prevent the form submitting
  return;
}
// OK to submit the form
submit.disabled=true;
submitted=true;
submit.className='disabled';
// this next bit is for demonstration purposes only
e.preventDefault();
```

26. Check to see what difference this has made.

If the form is invalid (username and/or password not OK) or the form has already been submitted, the Event Listener will prevent form submission by calling the forms `preventDefault()` method which cancels the default behaviour of the Event. Note we are using the `return` command to exit the function at this point.

If the form is valid and hasn't already been submitted, the button is disabled and we record that the form has been submitted. The normal form submit action would then occur – however we have prevented this, just so that we can see that this has occurred. The second `e.preventDefault()` call is just there so we can see the changes to the form, normally we would not include this line of code, the form would be submitted and we would no longer be able to see the form.

We are now going to take a look at some other form controls, starting with checkboxes.

Create and upload the following HTML file to the server. It uses the same CSS as the last HTML page, but a different JS file.

```
<!DOCTYPE html>
<html lang="en">
  <head>
```

```
    <meta charset="utf-8">
    <title>Form Two</title>
    <link rel="stylesheet" href="eventcss.css" />
  </head>
  <body>
    <div id="page">
      <h1>Course Catalogue</h1>
      <h2>Register Details</h2>

        <form action="FormEcho.php" method="post" id="details">

          <label><input type="checkbox" id="all"
value="all">All</label>
          <label><input type="checkbox" name="subject"
value="Computer Science">Computer Science</label>
          <label><input type="checkbox" name="subject"
value="Informatics">Informatics</label>
          <label><input type="checkbox" name="subject"
value="Computer Security">Computer Security</label>

          <input type="submit" id = 'submit' value="Save">

      </form>
    </div>
    <script src="events2.js"></script>
  </body>
</html>
```

We want our form to have the following functionality. If the user checks 'all' we want the individual subject check boxes to be checked automatically. If the user unchecks one of the individual subjects, then the 'all' checkbox is automatically unchecked.

Create and upload the following JS file to the server. Make sure it is called from your HTML.

```
(function(){
  var form = document.getElementById('details');
  var elements = form.elements;
  var options = elements.subject;
  var all = document.getElementById('all');

  function updateAll(){
    for (var i = 0; i < options.length; i++){
      options[i].checked = all.checked;
    }
  }

  all.addEventListener('change', updateAll, false);

  function clearAllOptions(e){
    var target = e.target;
    if (!target.checked){
      all.checked = false;
    }
  }

  for (var i = 0; i < options.length; i++){
    options[i].addEventListener('change', clearAllOptions, false);
  }
} ());
```

Activity

27. Check the behaviour of the checkboxes on the form.
28. Note that if the user checks all the individual subject checkboxes, the 'all' checkbox is not automatically checked.

Let's take a look at the JavaScript to see how this is working.

Firstly, note that everything in our JavaScript file is now wrapped into some sort of anonymous function:

```
(function(){
…
}());
```

Technically this is called an Immediately-Invoked Function Expression (IIFE). Just to be confusing, we can also write this as:

```
(function(){
…
})();
```

Note the difference in the parentheses in the last line. As far as we are concerned, either will do.

IIFEs have a number of important characteristics, but these really only matter when we become proficient JS programmers. For now the important things are that they are executed immediately and that their variables and internally defined functions are private – they cannot be accessed outside the scope of the function.

In these declarations:

```
  var elements = form.elements;
  var options = elements.subject;
```

we are first collecting all the elements in the form, then we are selecting all the elements whose `name='subject'`. This is in the form of an array.

In the `updateAll()` function, we are looping through the elements in the array, setting them to checked if 'all' is checked (true), or unchecked if 'all' is unchecked (false). This function fires if the value of the 'all' checkbox changes (goes from checked to unchecked, or vice versa) as indicated by the `'change'` event.

The `clearAllOption(e)` function sets the 'all' checkbox to false (unchecked) if the element (in this case a checkbox) that called the function is itself unchecked.

The final `for` loop loops through the array containing the `subject` checkboxes and attaches an Event Listener to each one. The event Listener fires if the checkbox value is changed. It calls the `clearAllOption(e)` function to check if the 'all' checkbox needs to be unchecked (in the case of a subject checkbox itself being unchecked).

Activity

29. We noted earlier that if the user checks all the individual subject checkboxes, the 'all' checkbox is not automatically checked. Modify the JavaScript to correct this problem.
30. Check your modified code behaves correctly – the 'all' checkbox should be automatically checked if all the subject checkboxes are checked. If the 'all' checkbox is checked, all the subject checkboxes should be automatically checked. If any of the subject checkboxes are not checked, then the 'all' checkbox should not be.

Next we will look at radio buttons. Create and upload the following HTML file to the server. It uses the same CSS as the last HTML page, but a different JS file.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Form Three</title>
    <link rel="stylesheet" href="eventcss.css" />
  </head>
  <body>
    <div id="page">
      <h1>Course Catalogue</h1>
      <h2>Publicity</h2>
      <form action="FormEcho.php" method="post" id="publicity">

        <label><input type="radio" name="source" value="print"
id="print">Newspaper or Magazine</label>
        <label><input type="radio" name="source" value="internet"
id="internet">Internet</label>
        <label><input type="radio" name="source" value="other"
id="other">Other</label>
        <input type="text" name="other-text" id="othertext">

        <input type="submit" id = 'submit' value="Save">

      </form>
    </div>
    <script src="events3.js"></script>
  </body>
</html>
```

We want our form to have the following functionality. The user can select "Newspaper or Magazine" or "Internet" or "other" – because we are using radio buttons, the user can only select one of these. If the user selects "other" then we want to display a text input box so that they can specify what the other source was. If the user subsequently selects a different option, we want to hide and clear the "other" text input box.

Add the following style rule to the CSS

```
.hide {display: none;}
```

We will use this to hide and unhide the text input box.

Create and upload the following JS file to the server. Make sure it is called from your HTML.

```
(function(){
  var form, options, other, otherText, hide;
  form = document.getElementById('publicity');
  options = form.elements.source;
  other = document.getElementById('other');
  otherText = document.getElementById('othertext');
  otherText.className = 'hide';

  for (var i = 0; i < options.length; i++){
    options[i].addEventListener('click', radioChanged, false);
  }

  function radioChanged() {
    hide = other.checked ? '' : 'hide';
    otherText.className = hide;
    if (hide) {
      otherText.value = '';
    }
  }
}());
```

Let's take a look at the JavaScript to see how this is working.

Firstly, note that we are using an Immediately-Invoked Function Expression (IIFE) again.

In these declarations:

```
form = document.getElementById('publicity');
options = form.elements.source;
```

we are first collecting all the elements in the form, then we are selecting all the elements whose `name='source'`, including the "other" radio button. This is in the form of an array.

```
other = document.getElementById('other');
otherText = document.getElementById('othertext');
otherText.className = 'hide';
```

we are then getting the "other" radio button , `id="other"`, and the text input box, `id="othertext"`. We are then setting the `class` of the text input box to `'hide'` so it will not be visible when the form initially displays.

The `for` loop loops through the array containing the `source` radio buttons and attaches an Event Listener to each one. The event Listener fires if the radio button is clicked. It calls the `radioChanged()` function to check if the visibility of the "other" text input box need to be changed (i.e. made visible or hidden).

In the `radioChanged()` function, we are making the "other" text input box visible if the `other` radio button is selected and hiding it and emptying it if a different radio button was selected. It uses a different form of conditional statement, the ternary operator.

A ternary operator has the following structure:

```
condition ? expression if true : expression if false;
```

So, this:

```
hide = other.checked ? '' : 'hide';
```

is equivalent to this:

```
if (other.checked) {hide = '';} else {hide = 'hide';};
```

The ternary operator is more compact, but also a bit more cryptic if you aren't familiar with it.

Activity

31. Modify the JavaScript so that there is a prompt "please specify" that appears with the text input box if the user selects "other".
32. Check that the code behaves as expected.
33. We will now extend this example slightly to look at text areas. Rather than use a text input box to allow the user to specify the "other", we will use a text area and we will allow the user to enter up to 140 characters. Replace the text input box with a text area:

    ```
    <textarea rows="5" cols="30" name="other-text"
    id="othertext"></textarea>
    ```

34. Add a `<span>` after the text area so that we can use it to display character count messages when we need to:

    ```
    <span id="textCount" class="hide"></span>
    ```

35. Add an Event Listener to the text area:

    ```
    otherText.addEventListener('blur', function() {if
    (otherText.value.length <= 140){textCount.className = 'hide';}},
    false);
    ```

    If the character count is less than 140, this will hide any character count messages when the text area looses focus. Otherwise the message will remain visible.

36. Add two more Event Listeners to the text area, one to fire on `'focus'` and one to fire on `'input'`. Both should call a new function called `updateCounter`.
37. Write a new function `updateCounter(e)` which will check the number of characters the user has entered into the text area (`e.target.value.length`). If this is more than 140 characters a warning should be displayed. Otherwise the number of characters remaining should be displayed. Use the textCount span to display the message:

    ```
        textCount.className = '';
    ```

```
        textCount.textContent = message;
```

38. Check your code to make sure it behaves correctly.

Finally, we will look at select boxes. Select boxes are more complex than other form controls and they have extra properties and methods associated with them. Create and upload the following HTML file to the server. It uses the same CSS as the last HTML page, but a different JS file.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Form Five</title>
    <link rel="stylesheet" href="eventcss.css" />
  </head>
  <body>
    <div id="page">
      <h1>Course Catalogue</h1>
      <h2>Choose Course</h2>
      <form action="FormEcho.php" method="post" >

        <label for="awardType">Award: </label>
            <select id="awardType" name="awardType">
              <option value="choose">Please choose an Award
type</option>
              <option value="degree">Degree</option>
              <option value="hnd">HND</option>
            </select><br>

            <label for="subject">Subject: </label>
            <select id="subject" name="subject">
              <option>Please choose an Award type first</option>
            </select>

        <input type="submit" value="Select">

      </form>
    </div>
    <script src="events5.js"></script>
  </body>
</html>
```

We want our form to have the following functionality. The user can select "Degree" or "HND" from the first select box. If they select "Degree", then the second select box contains the degree subjects. If they selected "HND" then the second select box contains the HND subjects.

Note that the HTML does not contain the Degree or HND subjects, we will add these using JavaScript, depending on whether they select "Degree" or "HND".

Add the following style rule to the CSS

```
label[for='awardType'], label[for='subject'] {
        display: inline-block;
        width: 4em;}
```

This will just align the labels and the select boxes nicely.

Create and upload the following JS file to the server. Make sure it is called from your HTML.

```
(function(){
      var type = document.getElementById('awardType');
      var subject = document.getElementById('subject');

      var degrees = {
            degcompsci: 'BSc Computer Science',
            degmmcomp: 'BA Multimedia Computing',
            deginfosec: 'BSc Information Security'
      };

      var hnds = {
            hndcomp: 'HND Computing',
            hndmm: 'HND Multimedia',
            hndinfosec: 'HND Information Security'
      };

      type.addEventListener('change', function(){
            if (this.value === 'choose') {
                  subject.innerHTML = '<option>Please choose an Award
first</option>';
                  return;
            }
            var subjects = getSubjects(this.value);

            var options = '<option>Please choose a subject</option>';
            for (var key in subjects) {
                  options += '<option value="' + key + '">' +
subjects[key] + '</option>';
            }
            subject.innerHTML = options;
      }, false);

      function getSubjects(awardType){
            if (awardType === 'degree'){
                  return degrees;
            } else if (awardType === 'hnd'){
                  return hnds;
            }

      }

}());
```

Let's take a look at the JavaScript to see how this is working.

Firstly, note that we are using an Immediately-Invoked Function Expression (IIFE) again.

In these declarations:

```
	var type = document.getElementById('awardType');
	var subject = document.getElementById('subject');
```

we are getting pointers to the Element Nodes holding the two select boxes.

In these declarations we are creating two Objects, one holding the degree courses and one holding the HND courses.

```
	var degrees = {
		degcompsci: 'BSc Computer Science',
		degmmcomp: 'BA Multimedia Computing',
		deginfosec: 'BSc Information Security'
	};

	var hnds = {
		hndcomp: 'HND Computing',
		hndmm: 'HND Multimedia',
		hndinfosec: 'HND Information Security'
	};
```

This an alternative, more compact way of specifying Objects. It consists of key: value pairs (note the colon), separated by commas and all contained in curly braces.

```
	{
	key: value,
	key: value,
	key: value
	}
```

So this:

```
	var degrees = {
		degcompsci: 'BSc Computer Science',
		degmmcomp: 'BA Multimedia Computing',
		deginfosec: 'BSc Information Security'
	};
```

Is the same as this:

```
	var degrees = new Object();
	degrees.degcompsci = 'BSc Computer Science',
	degrees.degmmcomp = 'BA Multimedia Computing',
	degrees.deginfosec = 'BSc Information Security'
```

The function at the end, `getSubjects(awardType`, returns the degrees Object or the hnds Object depending on the value it has been passed.

The Event Listener which is attached to the type select box fires when the value in the select box is changed (i.e. a different value is selected) on `'change'`. If the value of the type select box (`this.value`) is `'choose'` (i.e. no award has been selected), it sets the subject select box to read `Please choose an Award first`. If the value of the type select box is not `'choose'` then it calls the `getSubjects(awardType)` function with the value that has been selected (`this.value`). It then loops through the object that is returned, building a string containing all the options from the Object. It then sets the subject select box to contain the string (`subject.innerHTML = options;`). It is worth taking a look at the loop as there are a couple of new things to observe:

```
        for (var key in subjects) {
                    options += '<option value="' + key + '">' +
        subjects[key] + '</option>';
                }
```

Firstly we are using a `for in` loop. We can use this to loop through Objects and arrays where the order in which we access the contents is not important. The basic structure is:

```
        for (variable in object) {
                    …
                }
```

In our code we have included the variable declaration in the `for` statement. This is just a more compact form. We could have written it as:

```
        var key;

        for (key in subjects) {
                    options += '<option value="' + key + '">' +
        subjects[key] + '</option>';
                }
```

The `variable` (in our case `key`) is bound to each property in the Object (in our case `subjects`) in turn, for example `degcompsci`, and `object[variable]` (in our case `subjects[key]`) is bound to the value of that property, `'BSc Computer Science'`. So our loop would append `'<option value="degcompsci">BSc Computer Science </option>'` to `options` on this iteration.

Activity

39. Modify the HTML and JavaScript to do the following. If the user selects an HND subject, the following information is displayed, "An HND is a two-year full-time course." If they select a degree subject, the following information is displayed, "A Degree is a three-year full-time or five-year part-time course. Please select mode." A select box for mode is also displayed, with the options "Full-time" and "Part-time".
40. Check your modified code behaves correctly according to the following test plan.

Award	Subject	HND Message	Degree Message	Mode Select box
Unselected	Unselected	No	No	No
Degree	Unselected	No	No	No
Degree	Selected	No	Yes	Yes
HND	Unselected	No	No	No
HND	Selected	Yes	No	No

41. Check that your modified code behaves correctly when the user interacts, for example, select award, select HND, does HND message appear, deselect award, does Subject reset, does HND message disappear.

This workbook has provided a basic introduction to enhancing web forms using JavaScript Event Listeners. Event Listeners allow web pages to identify and react to user inputs. Combining Event Listeners with JavaScript and DOM manipulation allows us to create web pages that are interactive and to provide a better user experience. This Workbook has not covered all the details, so you may well need to do some additional reading or look up specific details when you need them.