

# Chapter 13. JavaScript 2: Event Handling

## Table of Contents

Objectives .....	2
13.1 Introduction .....	2
13.1.1 Event-based Programming .....	2
13.1.2 Event Handlers 'One Liners' .....	2
13.1.3 Events and objects .....	3
13.1.4 Anchor Events .....	4
13.2 Animating Button Images .....	7
13.3 Conditional Execution .....	9
13.3.1 JavaScript if statement .....	9
13.4 Code blocks .....	10
13.5 Boolean operators .....	11
13.6 General Selection .....	12
13.7 HTML Attributes for Event handling .....	13
13.8 Extension .....	14
13.8.1 Variables and their Scope .....	14
13.9 Review Questions .....	15
13.9.1 Review Question 1 .....	15
13.9.2 Review Question 2 .....	15
13.9.3 Review Question 3 .....	15
13.9.4 Review Question 4 .....	15
13.9.5 Review Question 5 .....	15
13.9.6 Review Question 6 .....	15
13.9.7 Review Question 7 .....	17
13.9.8 Review Question 8 .....	17
13.9.9 Review Question 9 .....	17
13.9.10 Review Question 10 .....	17
13.10 Discussions and Answers .....	17
13.10.1 Discussion of Exercise 1 .....	17
13.10.2 Discussion of Exercise 2 .....	17
13.10.3 Discussion of Exercise 3 .....	18
13.10.4 Discussion of Exercise 4 .....	18
13.10.5 Discussion of Exercise 5 .....	18
13.10.6 Discussion of Exercise 6 .....	19
13.10.7 Discussion of Exercise 7 .....	20
13.10.8 Discussion of Exercise 8 .....	20
13.10.9 Discussion of Exercise 9 .....	21
13.10.10 Discussion of Activity 1 .....	21
13.10.11 Discussion of Activity 2 .....	21
13.10.12 Discussion of Activity 3 .....	21
13.10.13 Discussion of Activity 4 .....	21
13.10.14 Discussion of Activity 5 .....	22
13.10.15 Discussion of Activity 6 .....	22
13.10.16 Answer to Review Question 1 .....	22
13.10.17 Answer to Review Question 2 .....	22
13.10.18 Answer to Review Question 3 .....	22
13.10.19 Answer to Review Question 4 .....	22
13.10.20 Answer to Review Question 5 .....	22
13.10.21 Answer to Review Question 6 .....	22
13.10.22 Answer to Review Question 7 .....	23
13.10.23 Answer to Review Question 8 .....	23
13.10.24 Answer to Review Question 9 .....	23
13.10.25 Answer to Review Question 10 .....	23

Answer to Review Question 9 .....	26
Answer to Review Question 10 .....	26

## Objectives

At the end of this chapter you will be able to:

- Write HTML files using JavaScript event handlers;
- Write HTML files using conditional statements and code blocks.

## 13.1 Introduction

The interesting behaviour of a system tends to be dependent on changes to the state of the system as a whole, or to its components. The kind of interaction a Web application might include usually involves short-term changes of state in which it is only important to know that they have occurred. That is the change of state is not intended to persist; it happens and it is not stored explicitly in the system. Such a change is indicated by an *event*. In the context of JavaScript, an event is an action that occurs in a browser that JavaScript provides facilities to detect and so act upon. Events are generally related to user interactions with the document, such as clicking and pointing the mouse, although some are related to changes occurring in the document itself. Programming JavaScript to handle such events provides for many styles of human-computer interaction. In short, programming JavaScript event handlers is crucial if you want interactive Web pages. When this style of programming dominates your design, it is known as *event-based programming*.

### 13.1.1 Event-based Programming

One event that you already know about occurs when the mouse is clicked on something, such as a hypertext link. Of course, the browser itself may intercept these events. You will note that many browsers change the status bar when the mouse is moved over an anchor. It is usually changed to the anchor's URL. In this case the browser has intercepted the event and has caused some action to occur. Events are useful for seeing what the user is doing and to provide them with extra information concerning their action.

Events are frequently used on forms to make it easier for the user to type in correct information, and to warn them when they input something incorrectly. For instance, if a text box requires a phone number, you can use events to notice whenever the user inputs data into the text box, and to ensure that the inputted data contains only numbers and dashes. Finally, you can validate all of the input before the user submits the form.

Events don't only have to be used to process forms. They could, for instance, be used when you have a number of frames which need to have their content changed when a user clicks on an anchor.

### 13.1.2 Event Handlers 'One Liners'

It is possible to add JavaScript to individual HTML tags themselves without using SCRIPT tags. These are often only single lines of code, and are thus nicknamed 'one liners'. This is not the only way to program event handlers, but is often the most convenient. This style of handling events is evidence of the close relationship between HTML and JavaScript: for a whole range of HTML elements tag attributes are provided that are associated with events. These attributes have as their value JavaScript code that is executed if the event occurs. For example, anchor tags support the event of a mouse pointer being moved over the anchor using the attribute *onMouseOver*. If a tag supports the event represented by the attribute, and the event occurs, then the JavaScript that is the value of the attribute is executed.

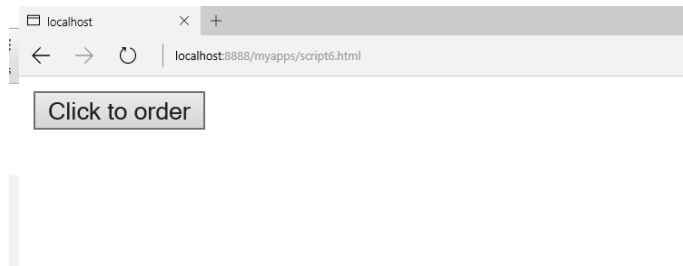
Many events can occur while a user is interacting with a Web page. For example a user might click on a button, change some text, move the mouse pointer over a hyperlink or away from one, and, of course, cause a document to load. There are event handlers for these events with names that include: *onClick*, *onMouseOver*, *onMouseOut*, *onLoad*. (We will be making use of all of these later.)

One of the simplest events is a mouse click. It is represented by the attribute *onClick* and supported by links and HTML button elements. Examine the following tag with an event handler as an attribute — a so-called 'one-liner'. (We will assume that this tag appears between *<FORM>* and *</FORM>* tags.)

## JavaScript 2: Event Handling

```
<INPUT type="button" value="Click to order"
onClick="window.alert('Purchase') ">
```

When a browser interprets this tag, it **renders** a button labelled *Click to order*. Subsequently, if a user clicks on that button, the click event is detected and the JavaScript associated with its attribute is executed. Here, an alert dialogue box is displayed, as shown below.



Let us work through this HTML and JavaScript. The first part of the `<INPUT>` tag is as you have previously seen: the *type* attribute is assigned the value *button*; the *value* attribute, which labels the button, is assigned the value *Click to order*. Then comes the new attribute for the tag `<INPUT>`. It is *onClick*, and is given the value that in this case is a single JavaScript statement that invokes the `window.alert()` method. This final attribute assignment creates an event handler for the particular JavaScript object representing the button such that clicking on the visual representation of the button causes the code to be executed.

In general, a sequence of statements may be included in the event handler. However, as it is essentially a 'one-liner'. Each line in the sequence must be separated by semicolons (as would be done in Java).

For example, including a second dialogue box that said 'Have a nice day' would require a semicolon, as in:

```
<INPUT type=button value="Click to order"
onClick="window.alert('Purchase window.alert('Have a nice
day') ">
```

This HTML/JavaScript works just as previously, except that clicking on the 'Click to order' button will produce a second alert box after the first has been dismissed.

### Exercise 1

Modify the earlier *onClick* example to include a flashing background colour before the alert dialogue box. Make sure you restore the initial background colour by saving it first with a variable and using the variable to restore the colour at the end. Depending on the speed of your computer, you will probably need at least two colour changes to notice anything.

You can find a discussion of this exercise at the end of the unit.

## 13.1.3 Events and objects

Earlier, it was suggested that you could conceive of the button as being an object with a nameless method that is invoked when you click on the button visible via a browser. You can, in fact, see that the HTML tag is implicitly creating a button object by accessing the button object and its properties - its type, as defined in the HTML tag, and its value, the text shown as the button label and defined by the VALUE tag. To do so the special variable *this* is used to refer to the object which the method belongs to. Hence, *this.type* can be used to access the type property, and *this.value* can be used to access the value property. The following variation of the first event handler can be used to confirm the object nature of the HTML button element:

```
<INPUT type=button name="orderButton" value="Click to order" onClick="windo
this.type + ' and has value: ' + this.value) ">
```

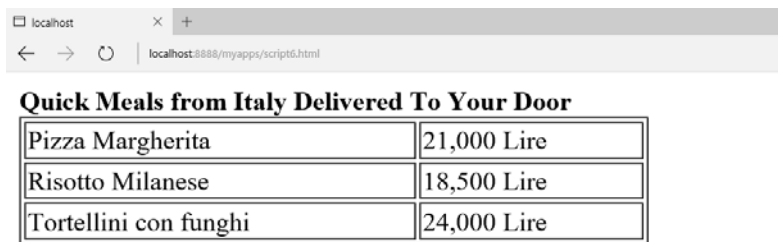
Executing this HTML `<INPUT>` tag (in a form) will produce the button as before, but when you click on it, the alert dialogue box now shows the two properties of the object referred to by *this*.

**Note**

Note that you can apparently change some properties of such an object. It is not clear that you would ever need to change the type of a button (e.g. from this sort of action button to a radio button) but you might want to change the label.

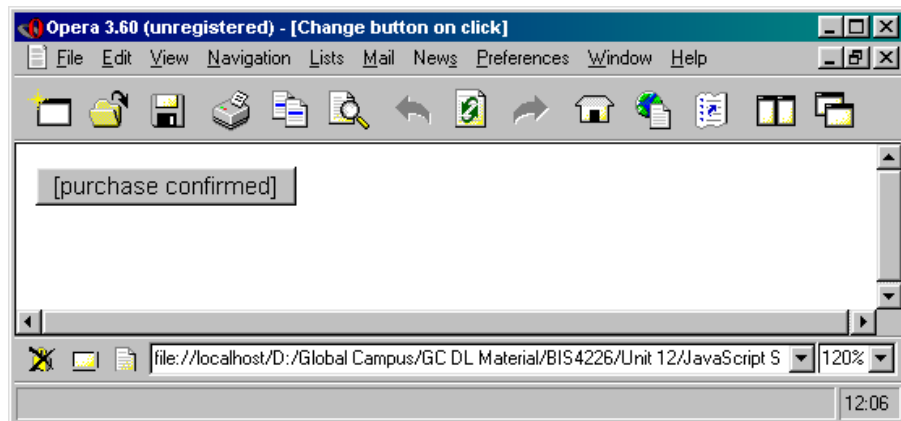
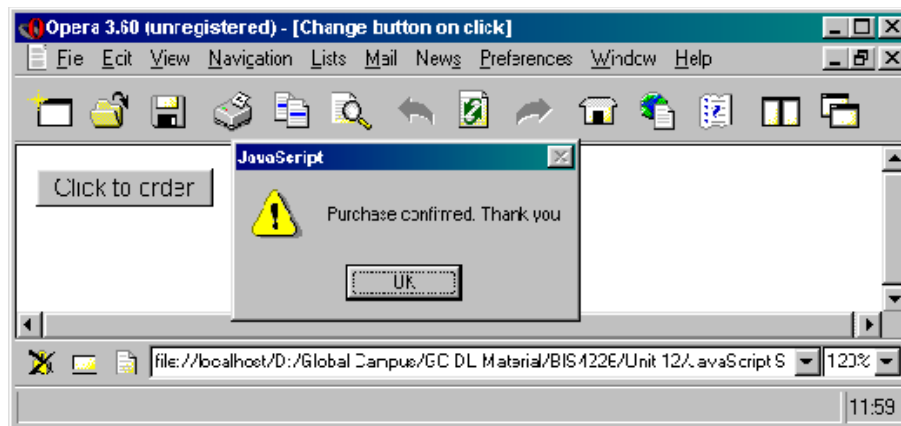
**Exercise 2**

Examine the original *onClick* example, which confirms a purchase, and the previous variation in which the button properties are accessed via the *this* keyword. Then devise HTML/ JavaScript that confirms a purchase, as in the original example of *onClick*, but which changes the label of the button after the confirmation to *[Purchase confirmed]*. See the diagrams below for the sort of thing you are aiming for.



Quick Meals from Italy Delivered To Your Door	
Pizza Margherita	21,000 Lire
Risotto Milanese	18,500 Lire
Tortellini con funghi	24,000 Lire

Hint: use the *this* keyword like a variable to assign a new string to the *value* property so that the new string is the text on the button.



You can find a discussion of this exercise at the end of the unit.

## 13.1.4 Anchor Events

As mentioned earlier, the anchor tag `<A>` can be enhanced to include JavaScript event handlers

## JavaScript 2: Event Handling

that correspond to the mouse pointer moving over the enclosed link, moving away from it and clicking on it. These anchor tag attributes are, respectively, *onMouseOver*, *onMouseOut* and *onClick*.

The general form is similar to that for *<INPUT>* with the event attribute following the link. Say you wanted to warn a user who had clicked on a link that the URL was not necessarily what they had wanted. For example, there is a UK company whose website URL is *www.apple.co.uk*. The company is not the UK division of Apple Computer Inc., so it might help a user to warn them what the link they had clicked on was maybe not what they wanted. (After the warning the user could stop the browser connecting to the server and go back.) The HTML/JavaScript is as follows:

```
<A href="http://www.apple.co.uk" onClick="alert('Remember this is not the Apple Computer Site')">Apple.co.uk</A>
```

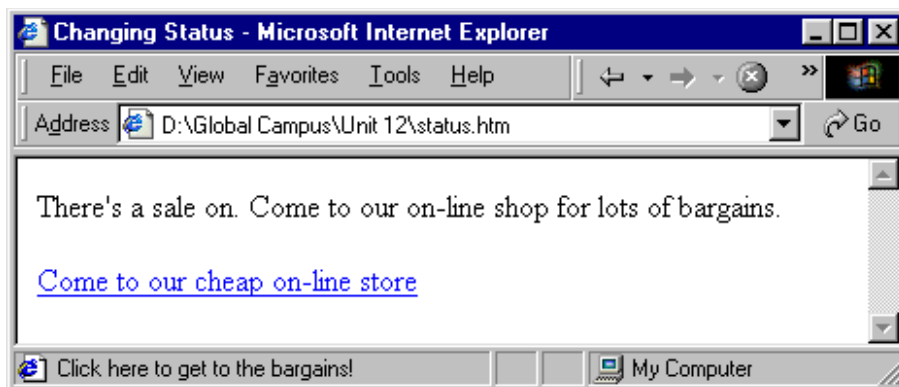
Of course warning a user after he or she has done something (clicked on the link) is not as helpful before one given before the action. The mouse-over event, which is programmed using the *onMouseOver* attribute allows this. For example:

```
<A href="http://www.apple.co.uk" onMouseOver="alert('Remember this is not the Apple Computer Site')">Apple.co.uk</A>
```

This tag differs from the previous one only by the replacement of *onClick* by *onMouseOver*. When the user moves the mouse pointer over the link, the dialogue box appears with its warning. Thus, the user can avoid the URL.

However, even this style is not optimal for the user. The warning can interfere with the interaction, requiring to be dismissed by clicking on OK or pressing the Enter key. A common practice is to use the window's status area, just as we did in the previous unit. We can avoid a browser's default behaviour of displaying a URL in the status area of the window's bottom bar, by inserting something more helpful to the purposes of the document — such as the kind of warning just discussed.

Let us take a different example, in which a document is meant to sell something to its user. You might to encourage the user to follow a link to some on-line shopping as soon as he or she moves over the link to the on-line shop, as in the following. First the document provides some ordinary text, followed by a link that reads ". Placing the mouse pointer over the link generates the text in the status area.



Note that the text 'Click here to get to the bargains!' only appears in the status area when the mouse pointer is over the link. This is achieved using the mouse-over event. As the name suggests, when the mouse pointer is over the link, the appropriate JavaScript code is executed. Here is what produces this interaction:

```
<P>
There's a sale on. Come to our on-line shop for lots of
bargains.
</P>

<P>
<AHREF = "http://www.most-expensive-sellers.com"
```

## JavaScript 2: Event Handling

```
onMouseOver = "window.status = 'Click here to get  
to the bargains!';return >  
</P>
```

### Exercise 3

Write down in your own words an explanation of what the above HTML and JavaScript in the anchor tag does.

You can find a discussion of this exercise at the end of the unit.

### Note

Note that the status area may not change back once the mouse pointer leaves the anchor, as this behaviour varies among browsers.

Strictly speaking there is something missing from the JavaScript of the *onMouseOver* event handler. It can be used for other actions than changing the status bar and it is useful for the browser to know whether the URL should be shown in status area (the usual behaviour) or not. For instance, in the first *onMouseOver* example the URL would still be shown in the status area after the alert dialogue box had been dismissed, especially if Enter had been used — because the mouse pointer would still be over the link. Including a *return true* statement at the end of the JavaScript 'one-liner' tells the browser that it should not display the URL. Although the status bar is to be occupied by our exhortation to come shopping, it is better to include the return value, as below:

```
<A HREF = "http://www.most-expensive-sellers.com"  
onMouseOver = "window.status = 'Click here to get to  
the bargains!'; return true;">Come to our cheap on-  
line store </A>
```

The use of a return value is a very important part of the above event handler. You will recall from our abstract object model (in Unit 10) that messages to objects can evoke a response. We encountered this with the *window.prompt* and *window.confirm* methods that return values. Many event handlers need to return a value for the browser to make use of. In the case in point, *true* is the response if we do not want the browser to display the URL in the anchor tag — exactly what is required here as we plan to change the status area anyway. However, if we want to see the URL, we must script the event handler to return *false*, as in the script below that changes the background colour but does not change the status area:

```
<A HREF = "http://www.most-expensive-sellers.com"  
onMouseOver = "document.bgColor = 'coral';  
return false">Come to our cheap on-line store</A>
```

As we have indicated, there is also an event that represents the mouse pointer leaving a link. This 'mouse-out' event is represented by the attribute *onMouseOut* and the code associated with it is executed when a user moves the mouse pointer away from a link.

### Exercise 4

Change the scripting in the previous anchor to restore background colour to what it was before being changed to coral. Hint: use a variable to remember the background colour property of the document's state. (You may find it convenient to look again at the discussion of variables in the previous unit.)

You can find a discussion of this exercise at the end of the unit.

The alternative version of the script in the previous exercise shows a common situation for which there is a shorthand notation. The situation is where a variable is declared and soon afterwards it is initialised, i.e. set to a first value, just as in the second version of the previous script.

## JavaScript 2: Event Handling

The shorthand allows initialisation at the same point as declaration, for example:

```
<SCRIPT>
var origBgCol = document.bgColor
</SCRIPT>
```

Be careful when using variables with handlers. As a general programming rule, you should declare variables close to where you use them. This might suggest that you should declare *origBgCol* in the handlers for both events, but this does not work because of rules of JavaScript. If handlers need to use the same variable (because they need to use the value the variable holds ) then declare the variable external to the handlers. (See the extension work on this topic)

### Activity 1: Clicking on Input Buttons

1. Implement the JavaScript of Exercise 1 to ensure you are familiar with the concept of handling an event using an HTML attribute.
2. Modify your implementation to prompt the user for his or her name, and then modify the button label using the button object's value property to include that name. With our current knowledge of JavaScript the button can only be modified in the event handler, so have it change after the first click.

Remember that if your PC or video card is very fast, you will probably not see the colours flashing.

You can find a discussion of this activity at the end of the unit.

### Activity 2: Programming Anchor Events

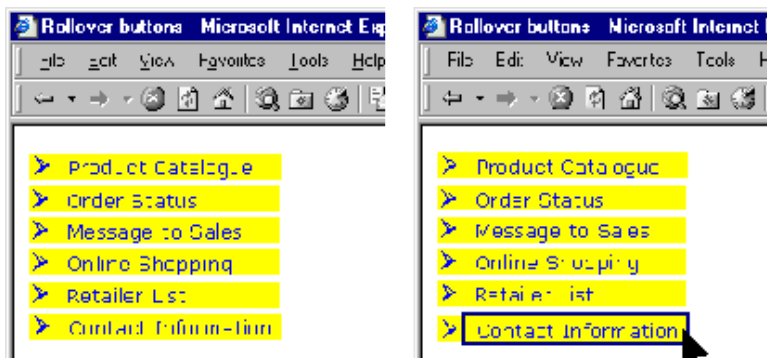
Change the HTML/JavaScript that exhorted you to go shopping for bargains (just before Exercise 4) so that when you click on the anchor the window status area changes to the greeting, 'Enjoy your shopping!' To test the JavaScript, *return false* from the handler to prevent the browser from following the link.

You can find a discussion of this activity at the end of the unit.

Now do Review Questions 1, 2 and 3

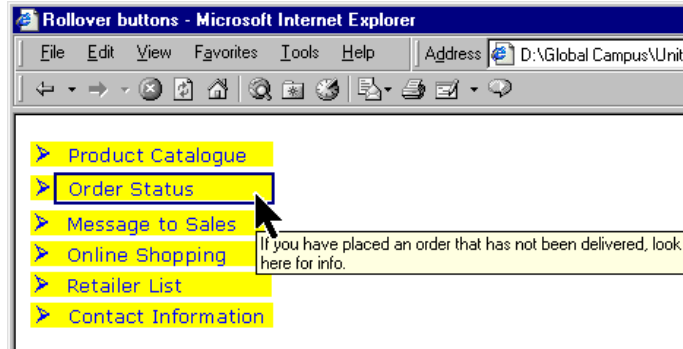
## 13.2 Animating Button Images

A common use for event handlers is to produce small animations to reinforce some aspect of the user interface. There are many situations where this may help the user to focus on some part of a Web page. For example, the buttons used in the screen shot on the left below can be enhanced by highlighting the button, as in the screen shot on the right below. This style of animation is known as a 'Rollover'.

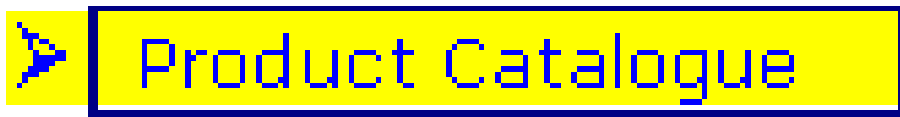
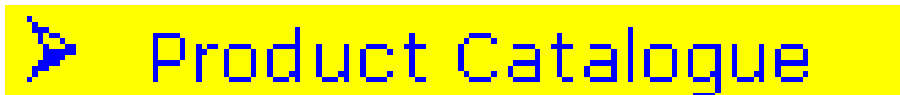


Indeed, most rollover arrangements will also include the text value of the images' *ALT* attributes, as in the diagram below:

## JavaScript 2: Event Handling



How can you script such an interaction? The basic strategy is to swap an image that represents an unselected button, for example the first image below, with an image that represents the selected button, such as the second image below, and then swap it back.



### Exercise 5

Can you think of the event or events that would trigger these swaps? And can you guess how you would refer to the image object from within the event handlers?

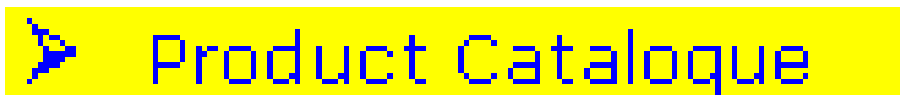
You can find a discussion of this exercise at the end of the unit.

The HTML/JavaScript code for the button rollover is as follows:

```
<IMG VSPACE=2 SRC="prod-cat-button.gif"
  onMouseOver="this.src='prod-cat-but
  onMouseOut="this.src='prod-cat-button.gif'"
  ALT="Check out the full Catalspecial orders.">
```

The *IMG* tag is just as you have previously seen. Ignoring the event handlers, it guarantees vertical spacing of two (via *VSPACE* attribute) and specifies the source of the image to be *prod-cat-button.gif* (via the *SRC* attribute); at the end of the tag it specifies the alternative (*ALT*) to the graphic image that also provides help to the user when the mouse pointer is over the button image.

As you might guess from the earlier use of the *this* variable with button objects, *this* is used in the event handlers to refer to the object corresponding to the graphical image being used as a button. As *IMG* tags have a *SRC* attribute, so to image objects have a *src* property accessible in JavaScript. Hence, within each of the event handlers the *src* property is changed to modify what image is being shown. In the mouse-over event handler, the image source is changed to refer to the graphic image that represents the selected version of the button. So, for example, if the image source specifies the file containing the first image below, moving over the image will change its source to specify the file containing the second image. When the mouse pointer leaves the image, the original graphic (image 1) is restored.







### Exercise 6

Complete the HTML/JavaScript for the button rollover interaction shown in the previous diagram using the 'this' style of referencing the image and placing break (<BR>) tags after each image (<IMG>) tag.

You can find a discussion of this exercise at the end of the unit.

### Activity 3: onMouseOver Event with an image

Choose a gif, jpg or png image of your choice. Write an HTML page that displays the image and produces a warning dialog box when the mouse pointer is over the image.

You can find a discussion of this activity at the end of the unit.

### Activity 4: onMouseOver Event with an image

Produce an HTML page that displays the image you used in Activity 3, but in this page make the status bar gives different messages according to whether or not the mouse pointer is over the image. You will need to use the event handler *onMouseOut*.

You can find a discussion of this activity at the end of the unit.

### Activity 5: changing an Image object

Every IMG HTML tag has an equivalent JavaScript image object. To make these objects easily accessible it is possible to provide a NAME attribute to the IMG tag. For example, to give an image the name "change", we would use the following IMG attribute:

```
NAME="change".
```

Now, in any JavaScript used in the HTML page, the image object will be stored in a variable called *change*. So, to access the image's src property, you could use the following code:

```
change.src
```

Now, try to set up an image and change it to another image when the mouse goes over it. If you have problems, look at the source code for the example above.

You can find a discussion of this activity at the end of the unit.

## 13.3 Conditional Execution

As in Java, JavaScript provides an *if* statement which, based on the value of a variable or other expression, can conditionally choose particular JavaScript code to execute.

### 13.3.1 JavaScript if statement

The *if* statement appears exactly as it does in Java (an expression that evaluates to *true* or *false* must be included in parentheses):

```
if (true or false expression)
statement
```

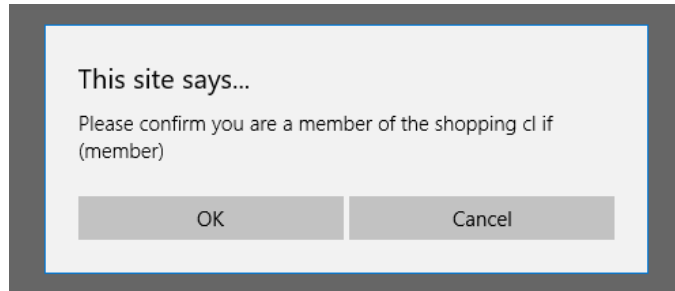
The expression in parentheses is evaluated and if it is true the following *statement* is executed; if it is false, it is not. Recall that the *window.confirm()* method returns *true* or *false* depending on

## JavaScript 2: Event Handling

whether the user clicks OK or Cancel. We will now use `confirm()` in an example of the *if* statement:

```
var member
member = window.confirm('Please confirm you are a member of
the shopping club')
if (member)
window.alert('Welcome to the Shopping Club')
</SCRIPT>
```

This code produces a confirmation dialogue box. When the user clicks OK a welcome dialogue box appears, as below.



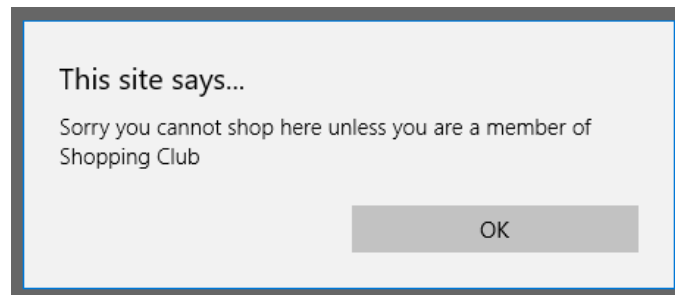
*if* statements also have a matching *else* statement, which behaves as in Java:

```
if(true or false expression)
  statement_1
else
  statement_2
```

`statement_2` is only executed if the expression in parentheses evaluates to false.

### Exercise 7

Change the previous script to display the following dialogue box if Cancel is clicked by the user when asked to confirm membership.



You can find a discussion of this exercise at the end of the unit.

## 13.4 Code blocks

To execute more than one statement in a conditional, you may use code blocks, which are sequences of statements packaged together between braces — `{` and `}`. Code blocks behave just as they do in Java, and can be used in the same places.

The most general form of an *if* statement can be written as follows:

```
if (true or false expression)
{
  statement_1a
```

```

    statement_1b
  }

```

and

```

if (true or false expression)
{
  statement_1a
  statement_1b
  statement_1c
  ...
  statement_1n
}
else
{
  statement_2a
  statement_2b
  statement_2c
  ...
  statement_2n
}

```

Let us extend the shopping example from the introduction to the *if* statement to provide the user with a table of possible purchases if they are a member of a shopping club.

### Exercise 8

Remembering that you can dynamically create the content of an HTML document using the *document.write()* method, change the shopping example to ask for confirmation of, say membership of an Italian food shopping club. If the user clicks OK, they should be presented with a table of shopping choices, as in the following:

#### Quick Meals from Italy Delivered To Your Door

Pizza Margherita	21,000 Lire
Risotto Milanese	18,500 Lire
Tortellini con funghi	24,000 Lire

You can find a discussion of this exercise at the end of the unit.

## 13.5 Boolean operators

So far we have used the *window.confirm()* method that is guaranteed to return either *true* or *false*. In a more general case, you will obtain true / false values by comparing two or more values.

Of course, you need to be able to compare all kinds of values and make a variety of comparisons. Here are the main operators for doing, which should be familiar to you from the Java module:

Operator	Example	Meaning
==	a == b	equality between any two values; returns true or false (example tests for a being equal to b)
!=	a != b	inequality between any two values; returns true or false (example tests for a not being equal to b)

Operator	Example	Meaning
===	a === b	identity between any two objects; returns true or false (example tests to see if a refers to the same object as b)
!==	a !== b	non-identity between any two objects; returns true or false (example tests to see if a does not refer to the same object as b)
<	a < b	less than between any two values; returns true or false (example tests for a being less than b)
<=	a <= b	less than or equal to between any two values; returns true or false (example tests for a being less than or equal to b)
>	a > b	greater than between any two values; returns true or false (example tests for a being greater than b)
>=	a >= b	greater than or equal to between any two values; returns true or false (example tests for a being greater than or equal to b)
&&	a && b	logical 'and' between two Boolean (true/false) values (example returns true only if both a and b are true, and false otherwise)
	a    b	logical 'or' between two Boolean (true/false) values (example returns false only if both a and b are false, and true otherwise)
!	!a	logical 'not' of one Boolean (true/false) value (example returns false if a is true and true if a is false)

## 13.6 General Selection

Because of the simple general form of the *if ... else* statement, you can use it as a very general selection mechanism by combining *if* statements in sequence, as in the code below, which determines the day of the week (in English) from the numeral for that day held by a *Date* object. Days of the week were numbered from 0 for Sunday, through to 6 for Saturday.

```
<SCRIPT>
var today = new Date()
var dayNo =
today.getDay() if (dayNo
== 0)
dayName = 'Sunday'
else if (dayNo ==
1) dayName =
'Monday' else if
(dayNo == 2)
dayName =
'Tuesday' else if
```

```

    (dayNo == 3)
    dayName =
    'Wednesday' else
    if (dayNo == 4)
    dayName =
    'Thursday' else if
    (dayNo == 5)
    dayName = 'Friday'
    else if (dayNo ==
    6)
    dayName = 'Saturday'
    window.alert('Local day
    is ' + dayName)
</SCRIPT>

```

Note that there are other control statements in JavaScript. An alternative to using multiple *if...else* statements would be to use a *switch statement*. Use your knowledge of Java to experiment with these statements.

### Exercise 9

*Navigator* objects provide properties and methods for manipulating the Web browser. A pre-declared variable called *navigator* can be used for this purpose. The following code writes out details of the browser:

```

document.write('appName = ',
navigator.appCodeName, '<BR>') document.write('appName
= ', navigator.appName, '<BR>')
document.write('appVersion = ', navigator.appVersion,
'<BR>') document.write('language = ',
navigator.language, '<BR>') document.write('platform =
', navigator.platform, '<BR>')

```

You can find a discussion of this exercise at the end of the unit.

### Activity 6: Remembering an Event

Modify the code of Exercise 2 (reproduced below), in which a button changed labels after being clicked, so that clicking the new button has no effect.

```

<FORM>
<INPUT type=button value="Click to order"
onClick="window.alert('Purchase confirmed. Thank
you'); this.value = '[purchase confirmed]'">
</FORM>

```

You can find a discussion of this activity at the end of the unit.

Now do Review Questions 7, and 8.

## 13.7 HTML Attributes for Event handling

There are many HTML attributes for handling events in JavaScript. The following table lists the main ones, and the objects that support them. You do not have to memorise this list, but you should be aware of the sort of events that can be handled.

HTML Event Attribute	Object supporting event
----------------------	-------------------------

HTML Event Attribute	Object supporting event
<i>onAbort</i> — triggered when document loading interrupted	<i>Image</i>
<i>onBlur</i> — triggered when an input element loses focus	<i>Text</i> elements, <i>Window</i> , other elements
<i>onChange</i> — triggered when user selects or deselects an element or enters text and moves focus to another input element	<i>Select</i> , text input elements
<i>onClick</i> — triggered when user clicks once; return false to cancel default action of following a link, submitting, etc.	<i>Link</i> , button elements
<i>onError</i> — triggered when an error occurs while loading an image	<i>Image</i>
<i>onFocus</i> — triggered when an element is given focus	<i>Text</i> elements, <i>Window</i> , other elements
<i>onKeyDown</i> — triggered when a key is pressed down by user; return false to cancel	<i>Document</i> , <i>Image</i> , <i>Link</i> , <i>Text</i> elements
<i>onKeyPress</i> — triggered when key is either pressed or released; return false to cancel	<i>Document</i> , <i>Image</i> , <i>Link</i> , <i>Text</i> elements
<i>onKeyUp</i> — triggered when the user released a key; return false to cancel	<i>Document</i> , <i>Image</i> , <i>Link</i> , <i>Text</i> elements
<i>onLoad</i> — triggered when document or image finishes loading	<i>Window</i> , <i>Image</i>
<i>onMouseDown</i> — triggered when mouse left button pressed down by user; return false to	<i>Document</i> , <i>Image</i> , <i>Link</i> , button elements
<i>onMouseOut</i> — triggered when mouse is moved away from element	<i>Link</i> , <i>Image</i> , <i>Layer</i>
<i>onMouseOver</i> — triggered when mouse pointer moved over element; for anchors, return true to	<i>Link</i> , <i>Image</i> , <i>Layer</i>
<i>onMouseUp</i> — triggered when user releases mouse button; return false to cancel	<i>Document</i> , <i>Image</i> , <i>Link</i> , button elements
<i>onReset</i> — triggered when form reset requested; return false to stop reset	<i>Form</i>
<i>onResize</i> — triggered when window is resized	<i>Window</i>
<i>onSubmit</i> — triggered when form submission requested; return false to stop submission	<i>Form</i>

Now do Review Questions 9 and 10.

## 13.8 Extension

### 13.8.1 Variables and their Scope

As you might have discovered, if you forget to declare a variable JavaScript will implicitly declare it for you. That is, if you write `origBgCol = document.bgColor`, JavaScript will declare the variable `origBgCol` if it has not previously been declared in some way (such as with a `var` statement). However, if you attempt to retrieve a value from a variable that has not been declared, as in `document.bgColor`

`= origBgCol`, then JavaScript will report an error.

Also associated with variable declaration is the variable's scope and its search chain. These define respectively

the region of the JavaScript program in which the variable can be seen, and the way in which JavaScript looks for the variable, and operate in the same way as they do in Java. Details can be found in *JavaScript, The Definitive Guide*, 3rd edition, by David Flanagan, O'Reilly Books, 1998. The rules explain why the following solution to Exercise 4 will not work:

```
<AHREF = "http://www.most-expensive-sellers.com"
  onMouseOver="var origBgCol=document.bgColor;
  document.bgColor = 'coral';return false"
  onMouseOut="document.bgColor=origBgCol">
Come to our cheap on-line store
</A>
```

In the above example, the scope of *origBgCol* is the *onMouseOver* event handler, as this is where the variable is declared. Since the *onMouseOver* and *onMouseOut* scopes do not overlap, when JavaScript searches for *origBgCol* in the *onMouseOut* handler will not find the variable, and since it is attempting to obtain a value from an undeclared variable, an error is reported. Declaring *origBgCol* in the *onMouseOut* handler would not work either, as all it would do is declare two separate variables which have the same name in different scopes.

The solution to this is to declare the variable outside of both handlers, in some scope which they can both access.

## 13.9 Review Questions

### 13.9.1 Review Question 1

An event takes place and is conveyed to the Web browser and thence to the document. Describe what it means to handle an event.

You can find the answer to this question at the end of the unit.

### 13.9.2 Review Question 2

Regardless of whether a handler has been provided, when a user of a Web browser clicks the mouse an mouse-down event occurs. True or false?

You can find the answer to this question at the end of the unit.

### 13.9.3 Review Question 3

Do all events have to be captured by JavaScript code?

You can find the answer to this question at the end of the unit.

### 13.9.4 Review Question 4

When do you return true from an anchor event handler?

You can find the answer to this question at the end of the unit.

### 13.9.5 Review Question 5

What is the special keyword that acts like a variable that you use to refer to an object in its event handler?

You can find the answer to this question at the end of the unit.

### 13.9.6 Review Question 6

## JavaScript 2: Event Handling

What is the basic strategy for animating button rollovers? You can find the answer to this question at the end of the unit.



## 13.9.7 Review Question 7

What is the name of the variable that refers to the browser object? You can find the answer to this question at the end of the unit.

## 13.9.8 Review Question 8

True or false: you can't capture a mouseOver event unless the mouse is over an anchor or an image. You can find the answer to this question at the end of the unit.

## 13.9.9 Review Question 9

Which is not an event handler attribute?

1. *onMouseClicked*
2. *onMouseOut*
3. *setTimeout*
4. *onLoad*

You can find the answer to this question at the end of the unit.

## 13.9.10 Review Question 10

True or false: Events occur on objects.

You can find the answer to this question at the end of the unit.

# 13.10 Discussions and Answers

## 13.10.1 Discussion of Exercise 1

The event handler starts the same with the HTML attribute *onClick*="... Then the variable for saving the initial background colour must be declared: *var currentBack*; with a semicolon terminating the statement. Then comes the colour changes, each with semicolons at the end of the statement. Next the initial colour is restored with *document.bgColor = curentBack*. A semicolon is needed after this too. The last statement is the *alert* message to *window*, as before. Putting it all together gives:

Note to implementer: the statement below must be on one line.

```
<INPUT type=button value="Click to order" onClick="var currentBack;
currentBack = document.bgColor; document.bgColor = 'blue';
document.bgColor = 'coral'; document.bgColor = 'blue';
window.alert('Purchase confirmed. Thank you')">
```

Note that if you were to try this on a fast PCs you might find that the flashing is not noticeable.

## 13.10.2 Discussion of Exercise 2

The HTML/JavaScript code is given below.

```
<FORM>
<INPUT type=button value="Click to order"
onClick="window.alert('Purchase
</FORM>
```

The JavaScript works as before: when the user clicks on the button a dialogue box is displayed. Then the label is changed using the assignment *this.value = '[purchase confirmed]'* (which is, of course, preceded by a semicolon).

But, what will happen if the user were to click on the newly labelled button? Exactly the same thing would happen. The handling of the previous *onClick* event has not been remembered — events are not remembered, they continuously happen — and so the behaviour of this little system remains the same. Later we will see how an event can, in effect, be remembered and that memory used to affect behaviour.

### 13.10.3 Discussion of Exercise 3

The first part of the tag and the text that immediately precedes the closing `</A>` tag are standard HTML and together set up a link to [Apple.co.uk](http://www.apple.co.uk) with the specific URL. What follows the value for the `HREF` attribute, the URL, is an attribute that specifies in JavaScript what reaction there is to be to an event that is expected to happen with this HTML tag. The expected event is that the user will place the mouse pointer over the anchor; this is denoted by the attribute *onMouseOver*. The value of the attribute (i.e. what follows the = symbol) is the JavaScript to 'handle' the event. The JavaScript simply changes the status area of the browser's border. So, the combination of the *onMouseOver* attribute corresponding to the mouse-over event, and the JavaScript to change the window status means that when the mouse pointer is placed on this link, the message string is displayed in the browser's status bar.

### 13.10.4 Discussion of Exercise 4

The JavaScript code is given below.

```
<SCRIPT>var origBgCol</SCRIPT>

<AHREF = "http://www.most-expensive-
sellers.com"
onMouseOver="origBgCol=document.bgColor;
document.bgColor = 'coral';return false"
onMouseOut="document.bgColor=origBgCol">
Come to our cheap on-line store<A>
```

The most important aspect of this code is the use of the *origBgCol* variable to remember the original background colour. The assignment takes place as part of handling mouse-over event, while the variable is declared *outside* the event handler. Alternatively, the assignment can also be done outside the event handler, thus:

```
<SCRIPT>
var origBgCol
origBgCol=document.bgColor
</SCRIPT>

<AHREF = "http://www.most-expensive-sellers.com"
onMouseOver="document.bgColor = 'coral';return false"
onMouseOut="document.bgColor=origBgCol">
Come to our cheap on-line store<A>
```

### 13.10.5 Discussion of Exercise 5

For the rollover style of interaction you need to swap the button image with the highlighted image when the mouse-over event is triggered, and swap it back when the mouse-out event takes place.

To refer to the image object whose image is to be swapped you need to use the *this* keyword.

## 13.10.6 Discussion of Exercise 6

Your code might look something like the one below. Note how *this* is used to denote the image object within whose tag the JavaScript appears.

```
<IMG VSPACE=2 SRC="prod-cat-button.gif"
  onMouseOver="this.src='prod-cat-button-sel.gif'"
  onMouseOut="this.src='prod-cat-button.gif'"
  ALT="Check out the full Catalogue with details of
    special orders.">

<BR>

<IMG VSPACE=2 SRC="order-status-button.gif"
  onMouseOver="this.src='order-status-button-
sel.gif'" onMouseOut="this.src='order-status-
button.gif'"
  ALT="If you have placed an order that has not been
    delivered, look here for info.">

<BR>

<IMG VSPACE=2 SRC="message-button.gif"
  onMouseOver="this.src='message-button-sel.gif'"
  onMouseOut="this.src='message-button.gif'"
  ALT="Submit a form requesting sales information or
    for someone to call you.">

<BR>

<IMG VSPACE=2 SRC="online-shop-button.gif"
  onMouseOver="this.src='online-shop-button-sel.gif'"
  onMouseOut="this.src='online-shop-button.gif'"
  ALT="Registered customers can browse products and
    prices and place orders online.">

<BR>

<IMG VSPACE=2 SRC="retailers-button.gif"
  onMouseOver="this.src='retailers-button-sel.gif'"
  onMouseOut="this.src='retailers-button.gif'"
  ALT="Find out about shops near you where you can
    examine and buy our products.">

<BR>

<IMG VSPACE=2 SRC="contacts-button.gif"
  onMouseOver="this.src='contacts-button-sel.gif'"
  onMouseOut="this.src='contacts-button.gif'"
  ALT="Names, postal addresses, telephone and fax
    numbers for you to contact all our departments.">
```

```
<BR>
```

## 13.10.7 Discussion of Exercise 7

```
<SCRIPT>
var member
member = window.confirm('Please confirm you are a member of the
shopping club')
  if (member)
window.alert('Welcome to the Shopping Club')
else
window.alert('Sorry you cannot shop here unless you are a member of Shopping
Club')
</SCRIPT>
```

## 13.10.8 Discussion of Exercise 8

The following JavaScript achieves the desired behavior. Notice the form of the *if ... else* statement and how code blocks are used.

```
<!DOCTYPE html>
<!-- Copyright (c) 2015 UCT -->
<html>
<BODY>
<SCRIPT>

var member
member = window.confirm('Please confirm you are a member of the Italian food
club')

if (member)
{
  //next are JavaScript document writes to include a table
  //of items and prices document.write('<TABLE BORDER=1>')
  document.write('<CAPTION><B>Quick Meals from Italy Delivered To Your
Door</B></CAPTION>')
  document.write('<TABLE style= "width: 40%" border = "1">')
  document.write('<TR>')
  document.write('<TD>Pizza Margherita</TD><TD>21,000 Lire</TD>')
  document.write('</TR>')
  document.write('<TR>')
  document.write('<TD>Risotto Milanese</TD><TD>18,500 Lire</TD>')
  document.write('</TR>')
  document.write('<TR>')
  document.write('<TD>Tortellini con funghi</TD><TD>24,000 Lire</TD>')
  document.write('</TR>')
  document.write('</TABLE>')
}
else
{
window.alert('Sorry, please go back to the membership enrolment page')
//next is JavaScript to go back to a membership form
document.write('<A href="http://www.food-
shopping.co.it/registration/register/"> Registration form </A>')
}
</SCRIPT>
</BODY>
</html>
```

## 13.10.9 Discussion of Exercise 9

```
if (navigator.appName.substr(0,8) == 'Microsof')

window.alert('This is a Microsoft browser')
if (navigator.appName.substr(0,8) == 'Netscape')
window.alert('This is a Netscape browser')
```

## 13.10.10 Discussion of Activity 1

Clicking on Input Buttons

1. The code from Exercise 1 is repeated below:

```
<INPUT type=button value="Click to order"
  onClick="var currentBack; curentBack = document.bgColor;
  document.bgColor = 'blue'; document.bgColor = 'coral';
  document.bgColor = 'blue';; window.alert('Purchase
  confirmed. Thank you')">
```

2. The modified version is given below.

```
<FORM>
<SCRIPT>
var person = prompt('What is your name?','')
</SCRIPT>
<INPUT type=button value="Click to order"
  onClick="var currentBack; curentBack = document.bgColor;
  document.bgColor = 'blue'; document.bgColor = 'coral';
  document.bgColor = 'blue';; window.alert('Purchase
  confirmed. Thank you '+person);
  this.value='Click to confirm, '+person">
</FORM>
```

## 13.10.11 Discussion of Activity 2

```
<A HREF = "http://www.most-expensive-sellers.com"
  onClick = "status='Enjoy your shopping!';return false">
Come to our cheap on-line store</A>
```

## 13.10.12 Discussion of Activity 3

Your code might look something like:

```
<IMG SRC="right.gif" onMouseOver="window.alert('Warning: over image')" ALT=
```

## 13.10.13 Discussion of Activity 4

Your code might look something like:

```
<IMG SRC="left.gif" ALT="Left image" onMouseOver='window.status="This image"'>
```

## 13.10.14 Discussion of Activity 5

Your code might look something like:

```
<IMG SRC="left.gif" onMouseOver="this.src='right.gif' "  
onMouseOut="this.src=
```

## 13.10.15 Discussion of Activity 6

The code is changed merely by adding a variable to remember that the originally labelled button was clicked and an *if* statement to test for this and avoid doing anything if the button has previously been clicked.

```
<FORM>  
<SCRIPT>var stillToConfirm = true</SCRIPT>  
<INPUT type=button value="Click to order"  
  onClick="if (stillToConfirm){window.alert('Purchase confirmed.  
  Thank you'); this.value = '[purchase confirmed]';stillToConfirm  
  = false}">  
</FORM>
```

## 13.10.16 Answer to Review Question 1

An event is something whose occurrence you can capture by providing JavaScript code to handle it in the object where the event takes place. When you capture it, you can execute JavaScript to perform some action.

## 13.10.17 Answer to Review Question 2

This statement is true. The event occurs whether there is a handler or not. If there is a handler the code is executed. A separate event occurs when the user releases the mouse button.

## 13.10.18 Answer to Review Question 3

No, most events are not captured by JavaScript code. The browser will handle events in a default manner if no code is written. Indeed most events are simply ignored by the browser.

## 13.10.19 Answer to Review Question 4

You return true when you do not want the browser to show the anchor's link URL in the window's status area.

## 13.10.20 Answer to Review Question 5

The keyword is *this*.

## 13.10.21 Answer to Review Question 6

You have to arrange that the graphics representing the normal button and the selected button replace each other as the mouse pointer is moved over the button (or anchor) and then away again. This means you have to update the *src* property of the image object to be assigned one graphic or another.

## **13.10.22 Answer to Review Question 7**

The browser object is referred to by the variable *navigator*.

## **13.10.23 Answer to Review Question 8**

This statement is false. There are many HTML tags that can have event handlers for the `mouseover` event. `<p>`, `<div>`, and `<h1>` and a few other marks.

## **13.10.24 Answer to Review Question 9**

`setTimeout` is not an event. It is merely a function which will cause an event to happen. Other functions and actions can cause events to happen. For instance, if a function loads a new document into the current window, this will cause an `onLoad` event.

## **13.10.25 Answer to Review Question 10**

This statement is true. An HTML mark such as `<a>` represents an object. In fact, there is a very close relation between HTML marks and JavaScript objects.