Topic support guide

Cambridge International AS & A Level

Cambridge International AS & A Level Information Technology 9626

For examination from 2017

Topic 19 Programming for the web

Cambridge Advanced

CAMBRIDGE International Examinations

Version 1.0

A DIVISION OF CAMBRIDGE ASSESSMENT

Contents

Introduction

How to use this guide

The aim of this guide is to facilitate your teaching of Cambridge International AS and A Level Information Technology, syllabus topic 19, Programming for the web. The guidance and activities in this resource are designed to help teachers devise programmes of study for using JavaScript to demonstrate object-based programming techniques and adding interactivity to web pages that provide teaching time devoted to theory work and opportunities for practical exercises.

Section 1 lists some key terms used in this topic and their definitions. Section 2 Theory provides an introduction to JavaScript programming techniques. Section 3 provides worked examples of programming tasks together with programming challenges to help learners understand some of the principles involved and the methods required to carry out practical tasks. Section 4 lists some further resources relevant to the topic for you or your learners to use. Section 5 provides theory questions and practical tasks that can be used as class or homework activities.

To write the JavaScript code we'll be looking at in this guide, you need a text editor and a web browser. All the files used and created in this guide are available within the zip file in the location from which you accessed this document.

Learning objectives

After reading this guide you should be able to teach the following learning objectives:

- demonstrate a range of object-based programming techniques
  - recognise data types (including: number, string, Boolean, array, object)
  - assign and understand the term variables
  - carry out calculations and basic string manipulation
  - use arrays
  - use comparison and logical operators
  - use conditional statements (including: if, else, else if, switch)
  - use loops (including: for, for/in, while, do/while)
  - use iterative methods
  - create functions
  - trap errors
  - control events
  - create html forms to interact with the user
  - add comments to explain JavaScript code
- add interactivity to webpages
- explain JavaScript terms and programming techniques.

Prior knowledge and preparation

Before you begin teaching this topic:

- Make sure you are familiar with the fundamentals of html and css.
- Make sure you understand the concepts and techniques shown in Section 2 Theory
- Undertake some further programming tasks using the resources suggested. This will provide depth to your knowledge and enable you to encourage learners to explore these topics further than the examples provided.
- Try carrying out all the tasks and develop some extension material or additional tasks.

1. Key terms

Word/phrase	Meaning
alert	Displays a box containing a message to the user / containing results
array	Arrays are indicated by square brackets and are used to store multiple values in a single variable, each of which can be accessed by referring to an index number. Array items are separated by commas.
bookmark	An identifier put into the body of the html code to specify where script results are placed in the webpage
Boolean	Booleans can only have either of two values: true or false.
conditional statement	Allows a program to perform different actions depending on the result of a condition; the most commonly used are: if, if…else, if…else if, switch
confirm	Displays a box that the user needs to click to continue / cancel
function	A block of code designed to perform a particular task that can be called anywhere in a script, and any number of times
global variable	Variable that can be used in functions other than the function in which it is declared or other parts of the script
local variable	Variable available only within the function in which it is declared
loop statement	Repeats a task until a certain condition is met; the most commonly used are: while, do…while, for, for…in
number	Numbers are one of JavaScripts data types. They can be written with or without decimals, and can use scientific notation for extra large or extra small numbers.
object	Objects are indicated by curly brackets and are containers for properties called named values. For example an object such as a blue bicycle would have a named value of colour:"blue". An objects properties use the following convention – name:value, name:value, name:value, etc.
operator	Symbol that indicates a specific process: arithmetic, string, assignment, comparison, logical, or conditional
pop-up	Alert, Prompt, Confirm boxes
prompt	Dialog box that asks the user for input and assigns the input to a variable
script	A high-level programming language (set of instructions), such as JavaScript that is 'interpreted' rather than compiled (e.g. when a web page is displayed)
string	A string is text like a letter, word or phrase. Strings are indicated by quote marks and either single or double quote marks can be used. Quote marks can be used within a string, but they mustn't match the quote marks that surround the string.
variable	Named container to which data is assigned; can contain the following data types: text (strings), number, boolean, array, object

2. Theory

2.1. Introduction

JavaScript is a scripting language that is primarily used for creating interactive features on webpages such as menus, form validation and changing images. It is the only language that is used to build applications on every platform: web, mobile, desktop, kiosk, TV etc.

As JavaScript is currently the only scripting language supported by every major web browser it is very widely used. When code is rendered (interpreted) by your web browser, as JavaScript usually is, it is called a 'client-side' script. Although this guide will address client-side scripting only, JavaScript can also be used as a very powerful server language, running as a 'server-side' script on a web server to generate html documents.

JavaScript is easy to learn. It has a straightforward syntax that is highly logical. It has fewer components than most other languages, and that makes it faster to learn. This introduction covers the basics – enough to get started with developing scripts to make web pages interactive. For every topic covered there are many more issues and options.

2.2. Comments

Comments are included by programmers to explain various sections of a script. Text after '//' is ignored by the browser until the end of the line. Larger comments on multiple lines can be commented out using the /*…*/ method as in CSS:

```
/* When more lines are
Needed for a longer
explanation of a section of the script*/
```

2.3. Alerts

Alerts are used to give the user messages, for example, if they did not fill in all the fields in a form. They are also a useful way of debugging the code: if a script doesn't seem to work, you can insert alerts to see how far the code gets processed before the error occurs. They can also be useful in displaying the value of a variable at important points. The syntax for an alert is:

```
alert ("User message");
```

Notice the ';' at the end of the statement. Statements are the sentences of JavaScript. Semicolons are the full stops. (Spaces and line-breaks are ignored.) The semicolons can sometimes be missed out but you should get in the habit of using them after every statement because omitting them can cause unexpected problems.

2.4. Variables

Like other programming languages, JavaScript uses variables. Variables are like named containers. You can place data into these containers and then refer to the data simply by referring to the name of the container.

Variables must be declared with the `var` keyword before they are used. (We'll see an exception to this rule in Section 2.7.)

Examples:

```
var price       var Price = "$2.00"     var length1=5;

var name        var Name = 'Pablo'      var length2=7;

var answer_1    var Answer_1 = 3.9      var Perimeter = 2*(length1 + length2);
```

Array variables:

```
var fruits = new Array("Apple","Banana","Cherry","Damson");
```
or
```
var fruits = ["Apple","Banana","Cherry","Damson"];
```

- Variable names are case sensitive, so 'price' and 'Price' are different variables
- Variable names must begin with a letter or an underscore character. For example, '1stvalue' is an invalid variable name but '_1stvalue' is a valid one
- The variables in the first column of the examples above are *undefined*, i.e. have no value assigned to them yet (the values are often assigned by user input)
- The variables in the second and third columns have been *initialised* – the values are assigned in the declaration
- Variables can be: strings (text), numbers, boolean (true/false), arrays or objects
  - Text strings need to be written inside double or single quotes
  - Numbers can be integers or decimals
  - Numbers inside quotes will be treated as strings
- There are a number of 'reserved words' that cannot be used for variables.
- You can assign a value to a variable using a calculation (as in `var Perimeter` above).

There is more information on variables in Section 2.7, and arrays in Section 2.10.

2.5. Getting user input – prompt boxes

One method to obtain input from users is to use a prompt dialog box. This presents a pop-up dialog on the webpage that prompts the user for the input. The user fills in the field and then clicks 'OK'. The user input is assigned to a variable. The syntax is:

```
var value1 = prompt ("explanatory text", 0);
```

- The variable is declared and the value set in one go.
- There are two parameters: the first parameter is the explanatory text that will be displayed and the second is the default value of the variable.
- Unlike other programming languages you don't have to tell JavaScript what type of value the variable will hold. JavaScript has 'dynamic datatypes', i.e. the type of a variable can change during the execution of a script and JavaScript takes care of it automatically.

2.6. Displaying output

To display output at a specific place on the webpage, put a 'bookmark id' into the body of the html code, where the output needs to be displayed:

```
<body>
 ...
<h2 id="Output here"></h2>
 ...
</body>
```

Theory

- The id= “Output here" bookmark is in the body of the html, not the script section so the html comment syntax is used
- The id is in style h2 so the result of the script will be displayed in style h2.

Setting the value of a `.innerHTML` property using the `document.getElementById`() method enables us to place results at a bookmark inside a webpage:

```
document.getElementById("Output here").innerHTML = 5 + 7;
```

2.7. Functions

A JavaScript function is a block of reusable code designed to perform a particular task. Functions can be called anywhere in a script, and any number of times. Creating a function means that the code does not have to be rewritten every time. Functions also allow programmers to divide long scripts into a number of small and manageable routines. We have already seen functions such as `alert()`. We can write our own functions as well.

Functions need to be defined. The syntax is:

```
function name (parameter1, parameter2, … ) { code statements to be executed; }
```

2.7.1 Calling functions

Functions do not automatically execute when the webpage is loaded or refreshed. Calling a function with a pop-up button therefore provides a useful method of activating scripts by interacting with the webpage. The button has to be displayed on the webpage, so it goes into the body section of the html:

```
<button type="button" onClick="name()">Click me</button>
```

- There are three types of button. This example uses `"button"`, the standard clickable button
- `onClick` will execute the `name()` function
- ‘Click me’ is the text displayed on the button.

2.7.2 Local and global variables

Variables that are available only within a function are *local* variables. Variables that can be used in other functions or other parts of the script are *global* variables. Global variables need to be declared **without** the ‘var’ keyword. Here, `value1` is declared as a global variable in `input()` so that it can be used by the function `output()`:

```
function input(){ value1 = 1;}
function output(){alert(value1);}
```

If `var value1` had been used, this would declare a local variable and `value1` would not be available for the `output()` function to use.

In fact, variables can always be declared in this way. You just need to be careful about whether you need local or global variables.

2.8. Operators

JavaScript has a number of types of operators. A full coverage of these operators is beyond the scope of an introduction but the following are some of the ones we might need most:

- Arithmetic operators
- String operators
- Assignment operators
- Comparison operators and Logical operators
- Conditional operator.

2.8.1 Arithmetic operators

Operator	Description	Example

+	Addition
	
	Note: JavaScript uses the '+' sign in two ways. If it thinks the variables are numbers, it will add them. Otherwise it will treat the variables as text and concatenate.
		`function multiply()`
		`{`
		`alert ("The answer is " + (value1 * value2))`
		`}`

- ***** **/**	Subtraction Multiplication Division	The use of these operators should be clear from the addition example.
++ **--**	Increment Decrement These operators are useful for counters in loops	The '++' operator adds 1 to the value of a variable. e.g. if valueX =15 then valueX**++** = 16 The – operator subtracts 1 from the value of a variable. e.g. if valueX=15 then valueX**--** = 14
%	Modulus (used for remainder or 'clock' arithmetic)	e.g. if valueX=15 then valueX%12 = 3 (15 mod 12= 3)

2.8.2 String operators

Operator	Description	Example
+	Concatenate This is the default mode for the '+' sign. Unless JavaScript thinks the variables are numbers it will concatenate the variables.	If valueX = "Hi" and valueY = "there" valueX + valueY = "Hithere" valueX + "" + valueY = "Hi there"
+=	Assign concatenated value to a variable	valueX += valueY assigns "Hithere" to valueX value X += "" += valueY assigns "Hi there" to valueX

Theory

2.8.3 Assignment operators

For these examples let's set valueX = 6 and valueY = 5

Operator	Description	Example
+=	Assign sum to a variable	valueX += valueY assigns valueX + valueY to valueX (valueX=11)
-=	Assign difference to a variable	valueX -= valueY assigns valueX - valueY to valueX (valueX=1)
***=**	Assign product to a variable	valueX *= valueY assigns valueX * valueY to valueX (valueX=30)
/=	Assign quotient to a variable	valueX /= valueY assigns valueX / valueY to valueX (valueX=1.2)

2.8.4 Comparison and logical operators

Comparison and logical operators return (produce a result of) true or false.
In the following examples let's set valueX = 6 and valueY = 5

Operator	Description	Example
== **!=**	Equal to Not equal in value	valueX == 6 returns **true** valueX == 5 returns **false** valueX != 5 returns **true** valueX == valueY returns **false** valueX != valueY returns **true**
=== **!==**	Equal value and equal type Not equal in value OR not equal in type	valueX === "6" returns **false** (because "6" is text) valueX !== "6" returns **true**
> >= < <=	Greater than Greater than or equal to Less than Less than or equal to	valueX > 6 returns **false** valueX >=6 returns **true** valueY < 5 returns **false** valueY <= 5 returns **true**
&& ||	Logical AND Logical OR	valueX <= 6 && valueY <= 5 returns **true** valueX < 6 && valueY <=5 returns **false** valueX >6 || valueY = 5 returns **true** valueX >6 || valueY < 5 returns **false**

2.8.5 Conditional operator

Operator	Description	Example
?	The conditional operator assigns a value to a variable based on a condition. The syntax is: `variablename = (condition) ? value1:value2`	In the following example, depending on whether a user guess is on target or not, they get a different message: `var result = (guess == target) ? "Brilliant! Good guess.":"Sorry your guess was wrong.";`

2.9. Conditional statements

Conditional statements allow a program to perform different actions depending on the result of a condition: a different block of code is run for each decision branch. In JavaScript there are four commonly used conditional statements:

`if`	Specifies a block of code to be executed, if a required condition is true
`if … else`	Specifies an alternative block of code to be executed if the same condition is false
`if … else if`	Specifies a new condition to test if the first condition is false
`switch`	Specifies alternative blocks of code to be executed

2.9.1 'if' statements

The 'if' statement is used to test for a certain condition, then do something (or not if the condition is not met). They are sometime known as 'if (then)': the 'then' is implied but not included in the code. They enable the program to behave differently depending on, for example, user input or some result within the script. The syntax is:

```
if (some condition is true) { execute these statements; }
```

- The statement starts with 'if' which must be lower case
- There is a condition in brackets ( ) which should give a true/false result
- There is a set of statements between curly brackets { } which will be executed if the condition is true
- Conditions can be combined using the logical operators 'AND' and 'OR' ('**&&**' and '||') that were explained in Section 2.8
- The spacing is not important – it's included for readability
- After the (condition), there is no ";".

2.9.2 'if … else' statements

'If … else' statements allow the program to run an alternative block of code if the condition is NOT met. The syntax is:

```
if (some condition is true) {  do something;  }
  else { do something else; }
```

Note that the 'else' part only runs if the first part (i.e. the 'do something') part does not.

2.9.3 'if … else if' statements

The 'if...else if' statement is just an extended form of 'if…else' that allows JavaScript to make a correct decision out of several conditions:

```
if (condition 1){statement(s) to be executed if condition 1 is true;}
  else if (condition 2){statement(s) to be executed if condition 2 is true;}
    else if (condition 3){statement(s) to be executed if condition 3 is true;}
     else {statement(s) to be executed if no expression is true;}
```

The code is fairly easy to understand. The relevant code statements are executed based on which condition returns true. If none of the conditions is true, then the final 'else' block is executed as a form of error trapping.

2.9.4 'switch' statements

The 'switch' statement is used to select one of many blocks of code to be executed, depending on the value of a variable. Using a single switch statement makes it easier to understand the logic of branching than using several 'if...else if' statements to create multiple branches.

JavaScript checks for each condition until a match is found. If nothing matches, a default condition will be used. The syntax is:

```
switch (resulting value of a variable)
  {
    case condition 1: statement(s); break;
    case condition 2: statement(s); break;
    …
    case condition n: statement(s); break;
    default: statement(s)
  }
```

- The `switch` expression is evaluated once
- The value of the variable is compared with the values of each *`case`*
- If there is a match, the associated block of code is executed
- If there is no match the `default` statement is executed
- The `break` statements indicate the end of a particular case. It is needed to stop the subsequent code from executing.

2.10. Loop statements

Loop statements repeat a task until a certain condition is met. Using a loop statement reduces the number of lines of code and complexity of the script when the same code needs to be executed multiple times, often with a different value for a variable(s). Loops are an extremely powerful aspect of programming and a good grasp of them is essential.

There are four important types of loop in JavaScript:

`while`	Loops through a block of code while a specified condition is true
`do … while`	Loops through a block of code while a specified condition is true; the code will be executed at least once
`for`	Loops through a block of code a specified number of times
`for … in`	Loops through the properties of an object

2.10.1 'while' loop

This is the simplest of the loop types. The 'while' loop keeps doing something *while* a certain condition is met. The syntax is:

```
while (some condition is true) { execute these statements; }
```

2.10.2 'do … while' loop

The 'do ... while' loop is similar to the 'while' loop except that the condition check happens at the end of the loop. This means that the loop will always be executed at least once, even if the condition is false:

```
do { statement(s) to be executed }  while (some condition is true);
```

2.10.3 ‘for’ loop

The 'for' loop is the most compact form of looping. It is a little trickier to understand, but is a bit more convenient to use and quicker to write. For the purposes of this introduction, the ‘for’ loop can be thought of like this:

```
for (initialise counter; test if condition is true; increment counter)
{ execute these statements; }
```

- The *loop initialisation statement* is executed before the loop begins, and sets the counter to a starting value
- The *test statement* tests whether a given condition is true or not. If the condition is true, the statements inside the loop will be executed, otherwise the loop will end
- The *iteration statement* increases or decreases the counter.

2.10.4 ‘for … in’ loop

The ‘for … in’ statement automatically loops through all the properties of an object or all the elements in an array without the need to initialise, test the value of, and increment the counter. For example, this code executes statements for each of the elements in the array ‘fruits’:

```
for (counter in fruits ) { execute these statements; }
```

- Each element in an array is numbered; the elements of the array are referenced by `counter`.

2.11. Getting user input – input boxes

Input boxes are a common method of collecting user responses. (We have previously seen the Prompt box method.) Input boxes are defined and positioned using html:

```
<body>
...
<input id="id" type="type"  value="initial value">
...
</body>
```

- The input boxes have three parameters: id, type, and initial value
- The type can be number, text, radio (button) or checkbox depending on the data required.

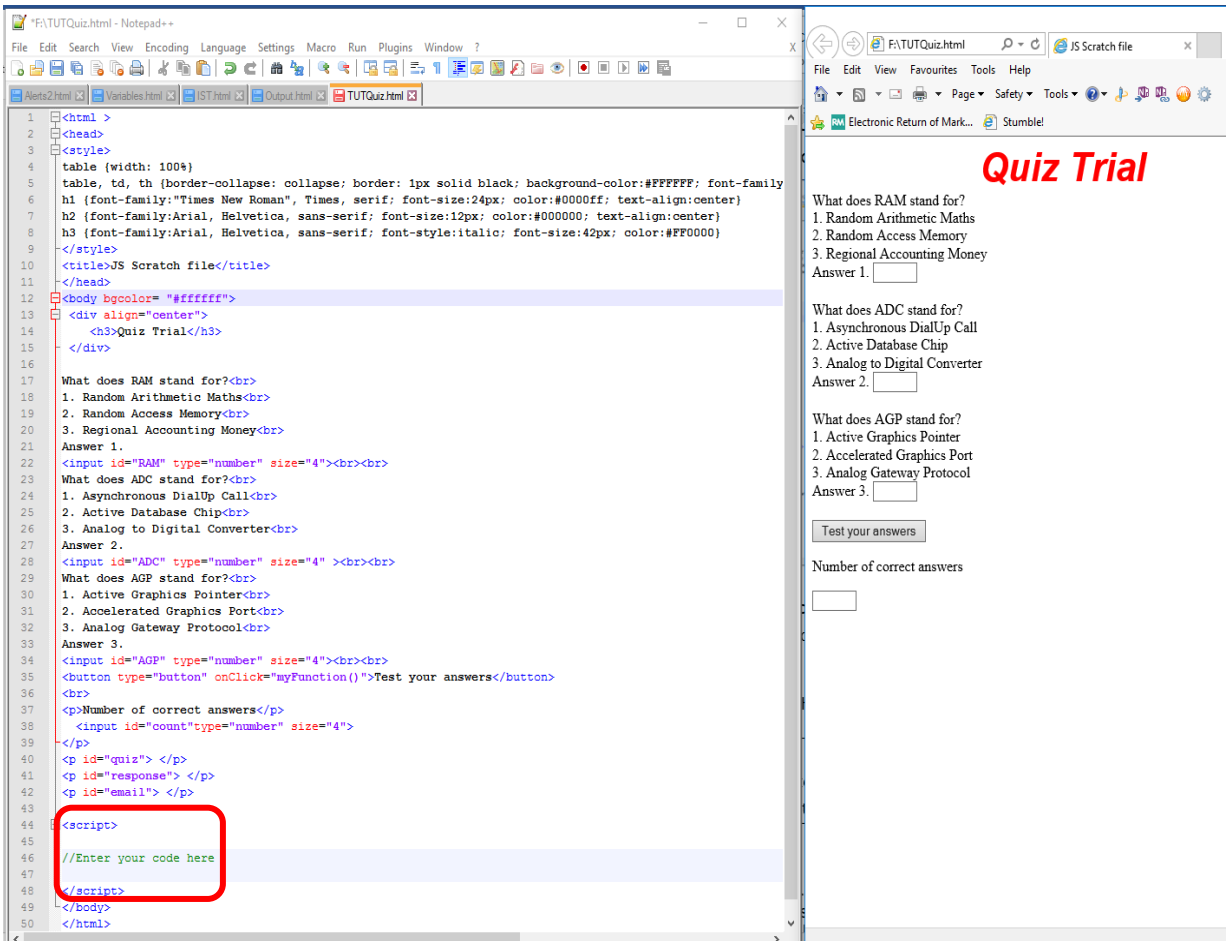
Note: html forms can be used as input boxes, but forms are really used for submitting the values to the web server for processing. In this topic, we are only looking at processing done by the webpage.

3. Worked examples

To introduce programming with JavaScript we'll look at short examples that can be tried and modified. This should help you understand the structure of a script and the effects of making changes. All the files used and created are available from within the zip file from which you accessed this document.

3.1. Developing scripts

The ideal way to experiment with running and modifying scripts is to have a text editor open in one window on your computer and a browser open in another. For the purpose of this tutorial the screenshots will show Notepad++ and Internet Explorer. This simple arrangement allows you to write and amend your JavaScript. Just resave the file and click the refresh on the browser toolbar. 



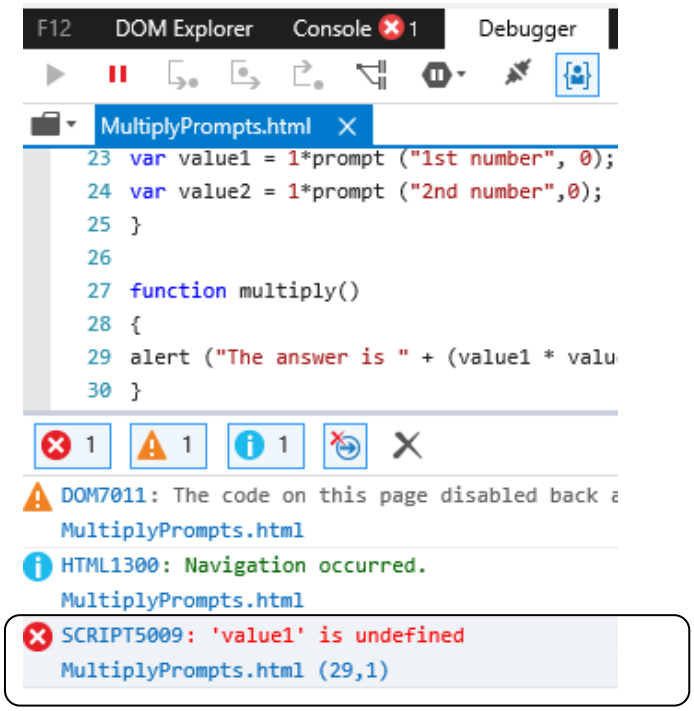
The screenshot shown is the html template for an online quiz. The code to make the quiz interactive can be added to the html between the <script> </script> tags. The <script> tag alerts the browser program to start interpreting all the text between these tags as a script.

It is quite usual to place the script between the </title> and the </head> tags but the script can appear anywhere in the <body> and <head> sections. It is preferable for the code to be located at the top or the bottom of the html as this makes it easier to find.

Note: You may see examples that use a type attribute <script type="text/JavaScript">. This is no longer required, as JavaScript is the default scripting language in html.

3.2. Debugging errors

When a program written in JavaScript has errors, we rarely see an error message to help in debugging. For example, when writing a script in stages we may see each stage work until the last addition, and then none of the stages are executed. However, most browsers have a Developer tools menu from which you can start a debugger. (Pressing F12 brings up the debugger in Internet Explorer, Firefox and Chrome.)



This is a view of the debugger for Internet Explorer. It tells us that the error that is stopping the script working is on line 29, and that the problem is that the 'value1' variable is undefined.

The script used 'var value1' and 'var value2' so these are defined as local variables, and are not available outside the function in which they are declared.

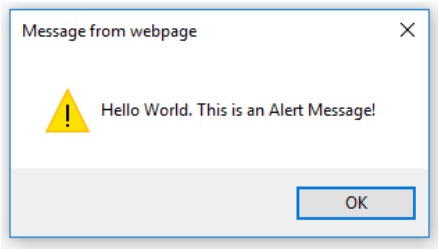
3.3. Alerts

We'll start with getting a webpage to display an alert box.

Enter the following code between the script tags in the file 'Blank_html_template.html':

```
<script>
//Display an alert box
alert ("Hello World. This is an Alert Message!");
</script>
```

Save the file as 'Alert.html'



Worked examples

Loading or refreshing the testing template webpage activates the alert automatically. Notice that you have to acknowledge the alert by clicking 'OK' before the script can continue or in this case conclude.

Programming challenge:

Try using multiple alerts like this:

```
alert ("Hello World. This is an Alert Message!");
alert ("Hello Internet. This the 2nd. Alert Message!");
alert ("Hello World Wide Web. This the 3rd. Alert Message!");
```

We will now look at what happens if you omit a semicolon at the end of a JavaScript statement. Try using the following on one line:

```
alert ("Hello World. This is an Alert Message!"); alert ("Hello Internet. This
the 2nd. Alert Message!");
```

In this case the browser sees two statements and the script works.

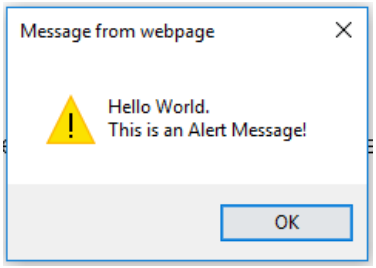
Now try with the first ';' missing:

```
alert ("Hello World. This is an Alert Message!") alert ("Hello Internet. This
the 2nd. Alert Message!");
```

In this case the browser sees only one statement and cannot interpret it.

Note: To display line breaks inside a popup box, use '\n':

```
alert ("Hello World.\nThis is an Alert Message");
```



3.4. Variables

Copy the following code into the <script> section of 'Blank_html_template.html', save the file as 'Variables.html' and load it in your browser.

```
<script>
var length1=5;
var length2=7;
var Perimeter = 2*(length1 + length2);
document.write(Perimeter);
</script>
```

Note: document.write() will place the value of the variable in the webpage.

You should see the answer '24' displayed. Note that we have no control over where the answer is displayed at the moment.

Experiment by changing the values assigned to `length1` and `length2`. (Remember to resave the file and refresh your browser each time.)

Study challenge:

- Find out what words are reserved in JavaScript
- Find the difference between a global variable and a local variable
- Investigate array variables.

3.5. Getting user input – prompt boxes

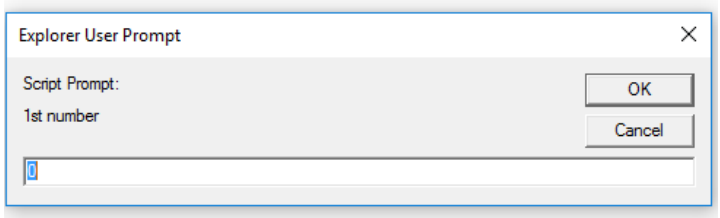
In the example above, the values used in the calculation are typed in the script. The following script uses prompt boxes so that the user can input values

Copy the following script into 'Blank_html_template.html' and save as 'Prompts.html':

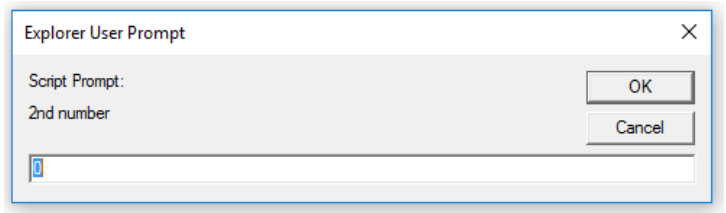
```
<script>
    var value1 = prompt ("1st number", 0);
    var value2 = prompt ("2nd number",0);
    document.write(value1 * value2);
</script>
```

Note: `document.write ()` can display the result of calculations.

When we load the page we should see the first prompt box:



Enter the number '7' and click 'OK'. We'll then see the second prompt box:



Enter the number '6' and click OK. The browser should display the answer 42 (7 x 6).

Now amend the script to:

```
<script>
var value1 = prompt ("1st number", 0);
var value2 = prompt ("2nd number",0);
document.write(value1 + value2);
</script>
```

Notice that the browser now displays 76! This is because the JavaScript default type for variables is text and the '+' sign is used to concatenate text. In the first example, JavaScript saw the '*' operator and so knew to multiply numbers. In the second example, JavaScript saw the '+' operator and used the default concatenate method.

We can 'trick' JavaScript into treating the '+' operator as the sum method.

Amend the script as follows:

```
<script>
var value1 = 1*prompt ("1st number", 0);
var value2 = 1*prompt ("2nd number",0);
document.write(value1 + value2);
</script>
```

Now Javascript has to treat the values as numbers and so adds them.

Study challenge:

Find out about the parseInt() and parseFloat() functions.

3.6. Displaying output

3.6.1 Bookmarks

Using the document.write() function as above is unsatisfactory. It is normally only used for testing purposes. We need to tell the browser where to place the results of our program by adding a bookmark id.

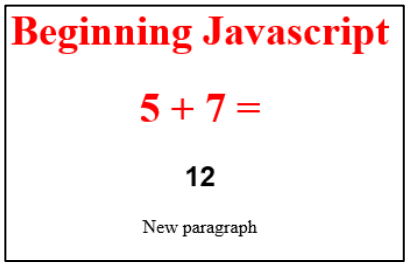
Look at the following extract of html:

```
<body>
<div align="center">
<h1>Beginning JavaScript</h1>
<h1>5 + 7 = </h1>
<!-- This is where we want the result of our script to be displayed -->
<h2 id="Answer here"></h2>
<p>New paragraph</p>
</div>
```

The following script will display the answer at the 'Answer here' bookmark:

```
<script>
document.getElementById("Answer here").innerHTML = 5 + 7;
</script>"
</body>
```

Copy the relevant parts of the code into 'Blank_html_template.html'and save the file as 'Outputs.html'. Load the page in your browser. The result of the script is now placed exactly where we specified:



Programming challenge:

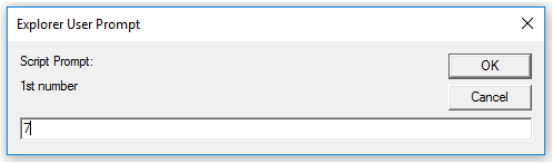
Experiment by moving the id bookmark to a new position – below the 'New paragraph' for example.

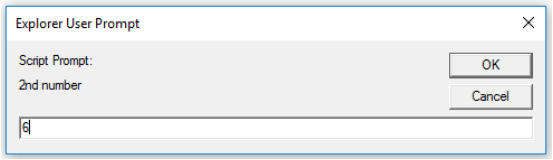
3.6.2 Alerts

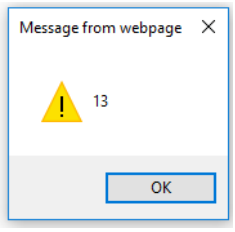
We can also display resulting variable values as an alert. Let's use the script from the 'Prompts.html' file again but replace the `document.write(value1 + value2)` statement with an alert.

```
<script>
var value1 = 1*prompt ("1st number", 0);
var value2 = 1*prompt ("2nd number",0);
alert (value1 + value2 )
</script>
```

Save the file as 'PromptsAlert.html', load the webpage and enter the values when prompted. The alert dialog displays the answer.

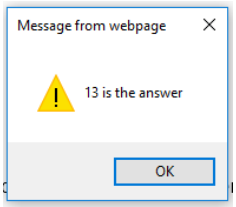






We can add text to the alert:

```
alert (value1 + value2 + " is the answer")
```



Programming challenge:

Try: `alert ("The answer is " + value1 + value2)`

You should now be able to explain why it doesn't give the correct result and be able to fix the problem.

3.7. Functions

All the JavaScript up to now has executed as soon as the webpage has loaded or been refreshed.

3.7.1 Functions example 1

Copy the following into the script section of 'Blank_html_template.html' and save the file as 'Greeting1.html'. We don't need any parameters for this function:

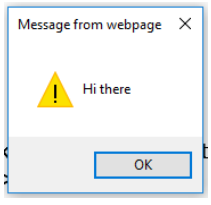
```
function Greeting() {alert ("Hi there");}
```

When you load the file nothing happens. We need to add a button to the webpage, in the body section of the html, to execute or call the function:

```
<body>
 <div align="center">
 <h1>Beginning Javascript</h1>
    <!-- This is will display a button we can click to execute the function -->
    <button type="button" onClick="Greeting()">Click me</button>
</div>
<script>
    function Greeting() { alert("Hi there");}
</script>
</body>
```

Resave the file and load it in your browser. You should see:

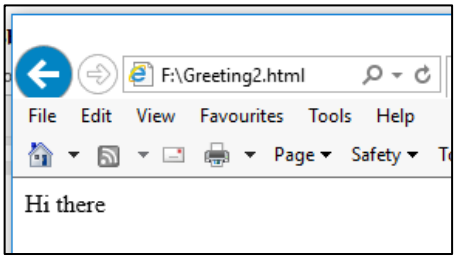
Clicking the button will display the message:



Because the button executes the `Greeting()` function you'll get the message every time you click 'OK' and 'Click me' again.

We have seen three ways to display results: alerts, document.write() and bookmarks (document.getElementById().innerHTML). Let's try using document.write(). Amend the script as shown and save the file as 'Greeting2.html':

```
<script>
function Greeting() { document.write("Hi there");}
</script>
```



This time we get just the greeting text – the original page contents are gone! There is no button to click. That's the big disadvantage of using `document.write()` in a function: it rewrites the entire page, losing the original html.

We'll have to use the `document.getElementById().innerHTML` method. We'll need a bookmark to tell the browser where to write the message. Let's put the bookmark under the button:

```
<body>
 <div align="center">
     <h1>Beginning Javascript</h1>
     <button type="button" onClick="Greeting()">Click me</button>
     <!-- This is where we want the message to be displayed -->
     <h2 id="Message"></h2>
</div>
```

Now we can amend the script:

```
<script>
    function Greeting()
    {
        document.getElementById("Message").innerHTML="Hi there";
    }
</script>
</body>
```

Note: This layout for the function is more common. It makes each component clear. Spaces and blank lines are ignored and a good text editor like Notepad++ prompts you with correct spelling and indicators to complete syntax such as closing the ( ) and { } brackets.

The function will now place the text 'Hi there' at the Message bookmark. Save the file as 'Greeting3.html' and load the webpage. This time, clicking the button displays the message within the webpage:

Repeated clicking doesn't appear to do anything, but it is in fact rewriting the message every time.

Worked examples

Let's try showing and hiding the message by clicking buttons. Amend the 'Greeting3.html' page as shown below and save the file as 'ShowHide.html':

```
<body>
 <div align="center">
     <h1>Beginning Javascript</h1>
     <button type="button" onClick="Show()">Show me</button>
     <button type="button" onClick="Hide()" > Hide me </button>
     <h2 id="Message"></h2>
 </div>
<script>
    function Show()
    {
    document.getElementById("Message").innerHTML="Hi there";
    }

    function Hide()
    {
    document.getElementById("Message").innerHTML=""
    }
</script>
</body>
```

The new `Hide()` function writes the `""` empty string at the Message bookmark when the 'Hide me' button is clicked.

3.7.2 Functions example 2

Let's go back and look at the 'PromptsAlert.html' file again:

```
<body>
 <div align="center">
    <h1>Beginning JavaScript</h1>
 </div>

<script>
    var value1 = 1*prompt ("1st number", 0);
    var value2 = 1*prompt ("2nd number",0);
    alert("The answer is "+ value1 + value2)

</script>
</body>
```

When the file is loaded in the browser the prompts to enter the numbers appear straightaway. This isn't very satisfactory. Let's write one function to initiate the prompts and another to multiply the values.

Let's also add two buttons to the html: one button to call the function that initiates the prompts to enter the numbers, and another to initiate the function that displays the alert showing the answer.

Here we add the two buttons in the body of the html:

```
<body>
 <div align="center">
     <h1>Beginning JavaScript</h1>
     <button type="button" onClick="inputs()">Enter the
     numbers</button>
     <button type="button" onClick="multiply()">Multiply the prompt
     values</button>
 </div>
```

Here we write the functions for each button to call. The variables `value1` and `value2` are declared as global variables so that they can be used in other parts of the script. Amend the file and save it as 'MultiplyPrompts.html':

```
<script>
    function inputs()
    {
    value1 = 1*prompt ("1st number", 0);
    value2 = 1*prompt ("2nd number",0);
    }

    function multiply()
    {
    alert ("The answer is " + (value1 * value2))
    }
</script>
</body>
```

Programming challenge:

This challenge revises what we have learnt so far.

Use the 'MultiplyPrompts.html' file and:

- amend the code to display the numbers entered in the prompts on the page
- amend the code so the answer is displayed on the webpage instead of using an alert
- amend the code so that the numbers and the answer show as an equation. e.g. 12 x 9 = 108.

3.8. Operators

Let's look at an example script for the conditional operator '?' which was introduced in Section 2.8. In this program, the user has to guess a number between 1 and 10. The correct answer is set to 8 and we use a prompt box for the user to enter a guess. Depending on whether the user guesses correctly, they will get a different message.

Worked examples

Copy the script into ‘Blank_html_template.html’. Save the file as ‘Guessgame.html’.

```
<script>
var target = 8;
var guess = prompt ("I’m an integer between 1 and 10\n Guess me", 0);
var result = (guess == target) ? "Brilliant! Good guess.":"Sorry your guess was
wrong.";
document.write(result);
</script>
```

Programming challenge:

Add a ‘Try again’ button to the html section of the ‘Guessgame.html’ file and amend the script to run when it is clicked.

Study challenge:

Investigate how to use the ! (Not) operator with the ‘greater than’, ‘less than’ operators.

3.9. Conditional statements

3.9.1 ‘if’ statements

Here's an example of an ‘if’ statement:

```
var question = prompt("Would you like to pause?", "Type yes or no");
if (question == "yes" { alert ("Paused. Click ok when you’re ready.");}
```

Note: The "==" operator in the condition means ‘equals’ as we’ve seen in Section 2.8. A common mistake is to use a single "=" which would mean that the variable ‘question’ is assigned the value ‘yes’ and so the ‘if’ condition will not work.

It is also worth noting that conditions can be combined using the logical operators ‘AND’ and ‘OR’, (‘**&&**’ and ‘||’) that are detailed in Section 2.8.

Here's an example of an ‘if’ statement that combines logical operators:

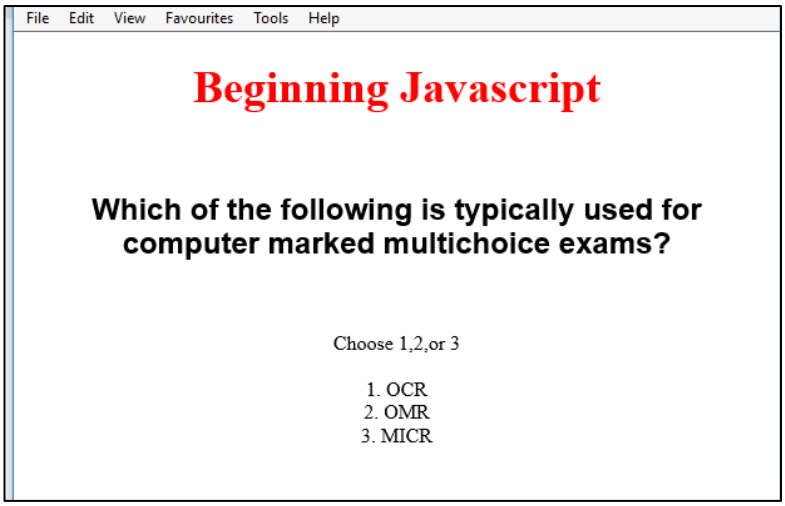
```
if (age >= 21 && age < 30) {statement to run if the condition is true};
```

The above specifies that the statement in curly brackets will run if the age is greater or equal to 21, AND less than 30.

3.9.2 ‘if … else if’ statements

Here’s a section of the html to display a webpage quiz:

```
<body>
 <div align="center">
    <h1>Beginning Javascript</h1>
   <h2> Which of the following is typically used for computer marked
   multichoice exams?</h2>
   <br><br>
   Choose 1,2,or 3<br><br>
   1. OCR<br>
   2. OMR<br>
   3. MICR<br><br>
```



Here's the script to make the page interactive. The <b><b>tags are just to make the text **bold**:

```
<script>
    var Answer = prompt("Choose your answer 1, 2 or 3",0);
    if( Answer == 1 ){ document.write("<b>No. OCR is for recognising Optical
    Characters</b>");}
    else if( Answer == 2 ){ document.write("<b>Yes. Optical Mark Recognition is
    correct</b>");}
    else if( Answer == 3 ){ document.write("<b>No. MICR is for recognising
    Magnetic Ink Characters</b>");}
   else{ document.write("<b>Unknown choice</b>");}
</script>
```

Copy the relevant html and script into 'Blank_html_template', save the file as 'if_else if.html' and open in your browser.

Programming challenge:

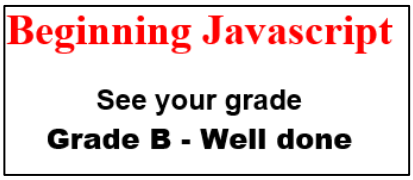
Customise the 'if_else if.html' page with your own alternative quiz with five choices and responses.

Here's another example of a script using 'if … else if' statements that returns the grade for an exam score. Copy the html and the script into 'Blank_html_template.html'and save the file as 'grades.html'.

```
<body>
 <div align="center">
 <h1>Beginning Javascript</h1> <br>
 <h2> See your grade<br>
<script>
    var Answer = prompt("Enter your score",0);

    if( Answer >=80 ){ document.write("<b>Grade A - Brilliant work</b>");}
    else if( Answer >=70 ){ document.write("<b>Grade B - Well done</b>"); }
    else if( Answer >=60 ){ document.write("<b>Grade C - Pass</b>"); }
    else if( Answer >=50 ){ document.write("<b>Grade D - Missed a pass</b>"); }
    else { document.write("<b>Fail</b>"); }
</script>
</div></h2>
</body>
```

Worked examples



Programming challenge:

Add in marks and comments for grades E (poor) and F (Fail).

3.9.3 'switch' statements

For an example of using the 'switch' statement, let's have another look at our last 'if._else if.html' quiz page. If we amend the script to use 'switch' it becomes much more simple.

Make the changes and resave the file as 'Switch.html':

```
<script>
    var Answer = prompt("Choose your answer 1, 2 or 3",0);
    switch(Answer)
    {
     case "1": document.write("<b>No. OCR is for recognising Optical
     Characters</b>");break;
    case "2": document.write("<b>Yes Optical Mark Recognition is
    correct</b>");break;
    case "3": document.write("<b>No MICR is for recognising Magnetic Ink
    Characters</b>");break;
    default: document.write("<b>Unknown choice</b>");
    }
</script>
```

If we try to amend the 'grades.html' file to use switch, however, we have problems with the numeric conditions (>=80 etc). 'If… else if' is sometimes less trouble.

Programming challenge:

Amend your new five choices quiz to use switch.

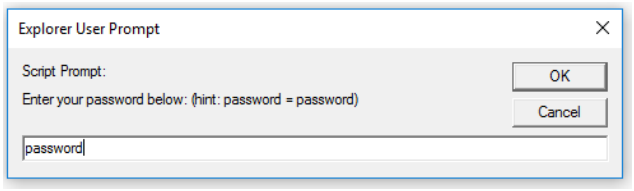
3.10. Loop statements

3.10.1 'while' loop

Here's an example of a 'while' loop that prompts for a password until the correct password is typed in. Copy the script into the 'Blank_html_template.html' and save the file as 'Password.html':

```
<script>
    //create variable
    var answer = "";

    while (answer != "password")
    {
    answer = prompt("Enter your password below: (hint: password =
    password)","");
    }
document.write("You're logged in")
</script>
```

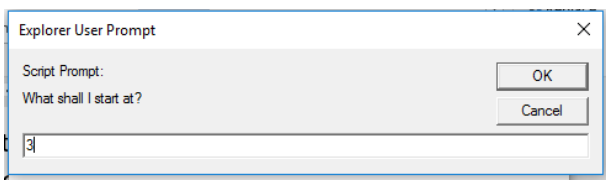


- The variable `answer` is declared with no value (`""`) before the `while` loop; this prevents a script error, as the `while` loop refers to the variable `answer` which has not yet been given a value from the `prompt ()`.
- The script states that `while` the response from the prompt is not equal (!=) to `password` it will always run the prompt again.
- When the variable `answer` does equal `password` the loop terminates and the script moves on to the `document.write("You're logged in")`.

3.10.2 'do … while' loop

Let's look at a harder example that counts up to eight. Remember that in 'do … while' loops, the code will be executed at least once, as the test is done after the 'do' statements. Copy the script into 'Blank_html_template.html' and save the file as 'do_while.html':

```
<script>
    var start_at = prompt("What shall I start at?",0);
    do {document.write(start_at+"<br>") ;start_at++}
    while (start_at <9);
</script>
```



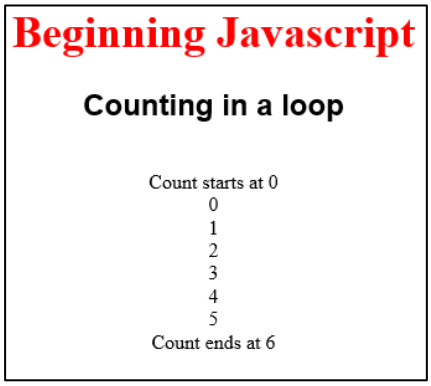
Programming challenge:

Create a prompt to set an 'end_at' variable to replace the '<9' condition.
Make sure the script displays the numbers up to **and including** the 'end_at' value.

3.10.3 'for' loop

Here's the script from 'forLoop.html' that produces the display shown when we enter '0' and '6' in the prompts:

```
<script>
    var start_at = prompt ("What shall we start at?",0);
    var end_at = prompt ("what shall we end on?",0);
    document.write("Count starts at " + start_at +"<br>");
    for (start_at; start_at < end_at; start_at++){document.write(start_at
    +"<br>");}
    document.write("Count ends at "+end_at)
</script>
```



Programming challenge:

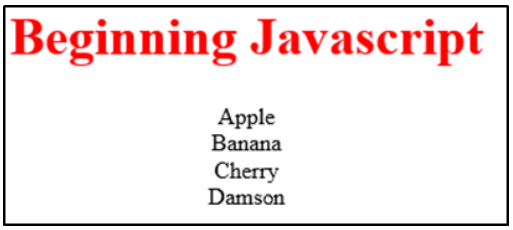
Amend the code to display the correct values as shown:

Count starts at 0
0
1
2
3
4
5
6
Count ends at 6

As the 'for' loop uses a counter we can count through the elements of an array. The elements in the array 'fruits' are numbered from 0 to 3.

This script uses the 'for' loop to write out each of the elements in the array on a new line:

```
<script>
    var fruits = ["Apple","Banana","Cherry","Damson"];
    var x;
    for(x = 0; x < 4; x++)
    {
    document.write(fruits[x] + "<br>" );
    }
    </script>
```



Note: In this script we used 'x<4' as the condition in the test statement. We had to count the number of items in the array to determine this value. We can get JavaScript to count the items in an array for us using: var x= fruits.length;

3.10.4 'for … in' loop

Let's compare the use of the 'for … in' loop, which automatically loops through all the properties of an object or all the elements in an array, with the 'for' loop we looked at above. This script from 'forInLoop.html' achieves the same results:

```
<script>
    var fruits = ["Apple","Banana","Cherry","Damson"];
    var x;
    for (x in fruits) {document.write(fruits[x] + "<br>");}
</script>
```

> Programming challenge:
>
> Use the array aquatics=["Alligator","Beaver","Caiman","Dolphin",”Eel","Flounder", "Gar"]; and amend the script to count the items in the array and set the condition in the test statement.

3.11. Displaying output – further practice with bookmarks

Apart from adding buttons and bookmarks, we have not looked at the html, but we need to spend a little more time to make sure we can use the document.getElementById().innerHTML method when needed.

For simplicity, until now we've displayed most output with document.write() as it made the code easier to understand. This is not good practice as we're not in control of where the output is placed on the page and we cannot format it. We need to specify where the output is displayed and be able to apply suitable formatting. (We also saw that using document.write() within a function rewrote the entire page, losing the original html.)

Let's revisit some of the examples above and display the output at a bookmark.
This is the original script for the password prompt box:

```
<script>
    //create variable
    var answer = "";
    while (answer != "password")
    {
    answer = prompt("Enter your password below: (hint: password =
    password)","");
    }
document.write("You're logged in")
</script>
```

Worked examples

This version has been amended to display the text ‘You're logged in’ at the bookmark ‘OutputHere’ in the body of the html under the title. The instruction to display the text ‘You're logged in’ is executed when the loop terminates. The text is in the html body so it can be formatted using style h2:

```
<body>
 <div align="center">
     <h1>Beginning Javascript</h1><br>
     <h2 id ="OutputHere"> </h2>
 </div>
<script>
    //create variable
    var answer = "";
    while (answer != "password")
    {
     answer = prompt("Enter your password below: (hint: password =
password)","");
    }
    document.getElementById("OutputHere").innerHTML="You're logged in"
</script>
</body>
```

Let's try another. Here's the script for the original quiz using ‘switch’:

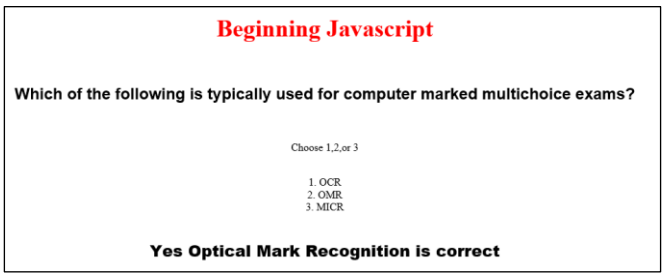
```
<script>
    var Answer = prompt("Choose your answer 1, 2 or 3",0);
    switch(Answer)
    {
     case "1": document.write("<b>No. OCR is for recognising Optical
     Characters</b>");break;
     case "2": document.write("<b>Yes Optical Mark Recognition is
     correct</b>");break;
     case "3": document.write("<b>No MICR is for recognising Magnetic Ink
     Characters</b>");break;
    default: document.write("<b>Unknown choice</b>");
    }
</script>
```

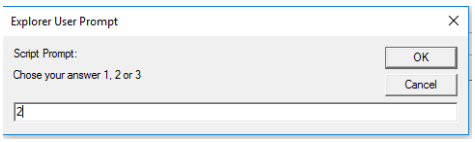
In order to display the resultant text where we want it and also to be able to format it we need to add the bookmark in the body of the html.

Here's the body of the html. You can see we've added the 'OutputHere' bookmark two lines (

) under the list of options and formatted it in style h2:

```
<body>
 <div align="center">
 <h1>Beginning Javascript</h1>
 <br>
 <h2> Which of the following is typically used for computer marked multichoice
exams?</h2><br><br>
 Choose 1,2,or 3<br><br>
 <p>
 1. OCR<br>
 2. OMR<br>
 3. MICR<br><br>
 </p>
 <h2 id="OutputHere"> </h2>
 </div>
<script>
    var Answer = prompt("Chose your answer 1, 2 or 3",0);
    switch(Answer)
    {
    case "1": document.getElementById("OutputHere").innerHTML="<b>No. OCR is
    for recognising Optical Characters</b>";break;
    case "2": document.getElementById("OutputHere").innerHTML="<b>Yes Optical
     Mark Recognition is correct</b>";break;
    case "3": document.getElementById("OutputHere").innerHTML="<b>No MICR is
     for recognising Magnetic Ink Characters</b>";break;
    default:  document.getElementById("OutputHere").innerHTML="<b>Unknown
    choice</b>";
    }
</script>
</body>
```





3.12. Getting user input – input boxes

This was the script 'Prompts.html' used to multiply two numbers:

```
<script>
    var value1 = prompt ("1st number", 0);
    var value2 = prompt ("2nd number",0);
    document.write(value1 * value2);
</script>
```

Worked examples

Like bookmarks, input boxes are defined and positioned using html. Here's some of the html with the additional input boxes. Open ‘Blank_html_template.html’ and add the following:

```
<body>
<div align="center">
<h1>Beginning Javascript</h1>
</div>
<p><br>Please enter the numbers here:<br>
<b>1st number:</b>
<!--1st input box -->
<input id="FirstNumber" type="number" value="">    
<b>2nd number:</b>
<!--2nd input box-->
<input id="SecondNumber" type="number" value="" >
</p>
```

Note: the use of the non-breaking space (‘ ’) above is to prevent the browser closing up the space between the first number entry and the second number entry.

Here's the script to carry out the multiplication:

```
<script>
    document.write(FirstNumber.value*SecondNumber.value)
</script>
```

The problem is that the script runs straightaway – before we get a chance to fill in the boxes. We need to execute the script on demand. For this we'll need to make the script a function (‘Multiply’) and assign it to a button like we did earlier.

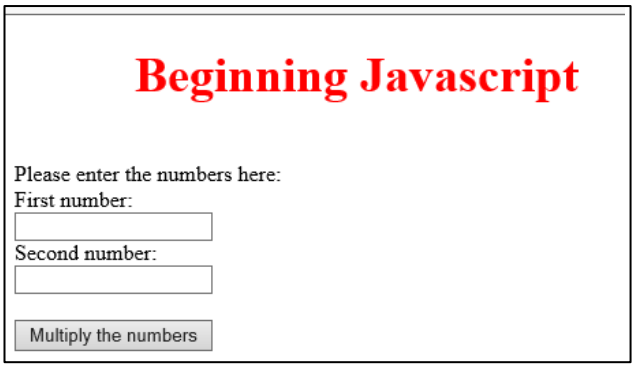
Here's the relevant part of the html:

```
<body>
 <div align="center">
 <h1>Beginning Javascript</h1>
 </div>
 <p>Please enter the numbers here:</p>
 <p>First number:<br>
 <input id="FirstNumber" type="number" value=""><br>
  Second number:<br>
 <input id="SecondNumber" type="number" value=""></p>
 <button type="button" onClick="Multiply()">Multiply the numbers</button>
 </p>
```

Here's the script using a function for the button to execute.

```
<script>
function Multiply()
{
document.write(FirstNumber.value*SecondNumber.value)
}
</script>
</body>
```

Now we'll see that the document.write() method overwrites the original html.



Save the script as ‘InputBox1.html’.

Programming challenge:

Use the document.getElementById().innerHTML method to display the result of the calculation on the web page.

Study challenge:

Compare the use of input boxes as seen above with the use of html forms.

4. Further resources

This introduction has covered most aspects of the syllabus, but learners are likely to need to carry out further study, as well as gaining more experience in programming. The following are recommended sites:

www.codeacademy.com/

A complete online course

www.w3schools.com/

A tutorial programme that can be also be used as a reference source

https://jsfiddle.net/

Provides a programming environment and a results window

https://notepad-plus-plus.org/

Notepad++ is a free download and has many features to speed up the development of scripts. It has an autocomplete facility to speed up code entry and avoid typos. It also highlights bracket pairs to make scripts more readable

The following are useful sites that provide online courses/tutorials:

www.tutorialspoint.com/javascript/index.htm
http://htmldog.com/
http://javascript.info/
www.learn-js.org/
www.javascriptkit.com/
www.homeandlearn.co.uk/JS/javascript.html
https://mva.microsoft.com/en-US/training-courses/javascript-fundamentals-for-absolute-beginners-14194?l=DmF3TY1eB_9500115888
www.echoecho.com/
www.howtocreate.co.uk/tutorials/javascript/

For specific issues and difficulties, use a search such as ‘how to use radio buttons’ to take you to examples and ‘try it yourself’ exercises.

5. Class and homework activities

5.1. Theory questions

1. What is the symbol(s) to comment out one line in JavaScript?

    a. //This is a comment
    b. **This is a comment
    c. %% This is a comment
    d. ** This is a comment **

2. How do you create a variable x that is equal to the string "Hello"?

    a. string x = "Hello";
    b. var x = "Hello";
    c. text x = "Hello";
    d. strings x = "Hello

3. Is JavaScript a case-sensitive language?

    a. yes
    b. no

4. How do you create a variable x that is equal to the integer 4?

    a. int x = 4;
    b. num x = 4;
    c. var x = 4;
    d. number x = 4;

5. The variable x is already defined and equals 4. Now, if the statement is "document.write(x++);", what would be displayed in the browser?

    a. 4
    b. 5
    c. 3
    d. 44

6. If x equals 7, and the only other statement is x = x % 3, what would be the new value of x?

    a. 7
    b. 3
    c. 4
    d. 1

7. Which of the following is not a comparison operator used in an if statement?

    a. ==
    b. !=
    c. *=
    d. <=

8. What is the keyword used inside a switch statement to create a condition?

    a. case
    b. condition
    c. if
    d. else

9. What is the keyword to define a function in JavaScript?

    a. def
    b. func
    c. function
    d. none of the above

10. Which of the following is an acceptable declaration of a for loop?

    a. for(var i = 0; i < 5; i++)
    b. for(var i = 0, i < 5)
    c. for(i < 5)
    d. for(var i = 0, i < 5, i++)

11. In a do while loop, which of the following is correct?

    a. The do statement is only executed if the while statement is correct.
    b. The while statement precedes the do statement.
    c. The do statement might not always be executed.
    d. The do statement is always run at least once.

12. What statement can you use to completely exit a loop and continue on through your code?

    a. break
    b. exit
    c. continue
    d. forward

13. What is a correct way to get an html element by its id?

    a. document.getElementsByID
    b. document.getElementsById
    c. document.getElementByID
    d. document.getElementById

14. How would you get the number of characters in the string variable example?

    a. example.length
    b. example.count
    c. example.len
    d. example.number

15. How do you create an array in JavaScript?

    a. var example = ('aa', 'bb');
    b. var example = array('aa', 'bb');
    c. var example = new Array ('aa', 'bb');
    d. var example = new array ('aa', 'bb');

5.2. Programming tasks

1. The solution for the ‘grades.html’ example is flawed.
   a) Amend the script so that it displays an error message when a number greater than 100 is entered.
   b) Add a button to allow another entry to be processed.

2. Open the ‘Quiz.html’ file in your browser. The ‘Test your answers’ button should call a function to count the number of correct answers.

   a) Add the necessary code to the file. The script should display the number of correct answers and the percentage correct at the bottom of the quiz.
   b) Amend the script to include five questions.
   Amend the script to add the text ‘Well Done’ if all questions were answered correctly or ‘Try again’ if not.
   c) Add a hyperlink to ‘Try again’ to reload the page.

3. Open the ‘Widgets.html’ file in your browser. The ‘Review Order’ button should call a function named ‘Acknowledge()’ to display the details of the order and the cost of the order in a new window.