Introduction to JavaScript

Resources

This tutorial was prepared using the following resources:

http://www.w3.org/

http://www.ecmascript.org/

http://www.w3schools.com/js/

http://www.w3.org/community/webed/wiki/Main_Page

Gosselin, D., 2011. JavaScript. 5th ed. Boston, MA: Course Technology, Cengage Learning.

Introduction

HTML Template

```
<!DOCTYPE html>

<html lang="en">

<head>
  <meta charset="utf-8" />
  <title>Introduction to JavaScript</title>
</head>

<body>
  <h1>Introduction to JavaScript</h1>
  <h2>Output from Exercises</h2>
  <div id="content">
    <span id="jsOutput">To be replaced with JavaScript output.</span>
  </div>
</body>

</html>
```

The script examples shown in this document will assume the above HTML as a base. Some examples may require additional HTML in the *<head>* or *<body>* element. Where this is necessary, the amended section will be included in the example.

This document is a brief introduction to JavaScript that assumes familiarity with a programming language such as Java. The tutorial will highlight JavaScript syntax through examples but will not detail programming concepts such as the use of variables, loops or branching structures such as *if* statements. The tutorial will also look at the Document Object Model (DOM) and how to use JavaScript to manipulate it.

While JavaScript may be used within the HTML document the best way to use JavaScript with your HTML mark-up is in an external file and using what is referred to as ‘unobtrusive

JavaScript'[1]. Examples of JavaScript will therefore be shown separate to the HTML. While this may seem laborious for some of the more simple examples shown, there are advantages in doing so. These advantages include accessibility, reduction of development cost/time/rework (for larger projects) and reduction of compatibility issues.

Things to Know

- JavaScript is also known as ECMAScript.
- JavaScript is case-sensitive.
- JavaScript is placed within the HTML *<script type="text/javascript">* element
- JavaScript is a sequence of statements to be executed by the browser. The statements will be executed in the sequence in which they occur.
- According to the ECMA specification, the inclusion of a semi-colon ';' to terminate a JavaScript statement is optional. However:
    - It is usually regarded as good programming practice to include it anyway.
    - In its absence, the browser should interpret a new line as the end of the statement.
    - Inclusion of the semi-colon means that multiple JavaScript statements may appear on the same line.
- Statements may be associated in blocks by enclosing them within braces '{' and '}'. For example:

```
{
 var mySpan = document.getElementById('jsOutput');
 displayEl.textContent = "Hello World";
}
```

  Blocking statements in this way becomes useful when they are grouped in a function or when they are to be executed depending upon a conditional state, such as may be found in an *if* statement.
- Commented lines will not be executed.
    - Single line comments are denoted by a double forward slash '//'. Anything typed after the single line comment is ignored until a new line has been reached.
    - Multiple lines may be commented out by enclosing them between '/*' and '*/'.

Hello World

Create the HTML

Create a file called "JSIntroHelloWorld.html". Copy and paste the HTML template (given in the Introduction section) as the file contents. Make the following amendment to the mark-up.

```
<head>
  <title>Introduction to JavaScript</title>
  <script type="text/javascript" src="HelloWorld.js"></script>
</head>
```

The *<script>* element can either contain script (if you are using script within your HTML mark-up) or it can be empty if an external script file is being used.

[1] http://www.w3.org/community/webed/wiki/The_principles_of_unobtrusive_JavaScript

The beginning tag of the <*script*> element contains the *type* attribute that specifies the MIME type of the script being used (in this case JavaScript). This attribute is required and has no default value, a value must be supplied.

The *src* attribute specifies the URL of the external script (in this case it will expect the file “HelloWorld.js” to be contained in the same folder).

When you are using an external script file, you can consider it as if the contents of the external file were to replace the <*script*> element within your HTML.

Create the JavaScript

Create a file called “HelloWorld.js” and save it with the following code.

```
function initialise()
{
 var mySpan = document.getElementById('jsOutput');
 mySpan.textContent = "Hello World";
}
window.onload=initialise;
```

The JavaScript above contains a function called *initialise()* which is called as soon as the HTML page is loaded into memory. When the page is ‘loaded’ it fires an ‘onload’ event. JavaScript can ‘listen’ for that event and run some code when it happens. In this case (using *window.onload=initialise;*) it calls the *initialise()* function.

Note: Functions will be covered later in the tutorial. For now, consider them as statement blocks that can be called multiple times without needing to re-type all the statements every time.

The *initialise()* function runs two lines of code. The first assigns the element in the HTML defined by *id=”jsOutput”* to the variable *mySpan*. Any changes made to the variable object *mySpan* will be made to the element in the HTML to which it refers. The second line of code in the function takes advantage of this by setting the value of the *textContent* property of the *mySpan* object to the string “Hello World”.

Run the page in a browser of your choice to see what happens.

Make the following amendment to “HelloWorld.js”.

```
function initialise()
{
 var mySpan = document.getElementById('jsOutput');
 mySpan.innerHTML = "<h1>Hello World</h1>";
}
window.onload=initialise;
```

Note the use of the *innerHTML* property instead of *textContent* and the inclusion of the HTML within the string.

Run the web page again to see what happens. If you were to keep *textContent* as the property of *mySpan*, what do you think the output would be? Try it and see.

Note that *textContent* is not supported by all browsers. It will work in the current versions of (at least) Firefox, Chrome and Opera. The alternative is *innerText* which will work in at least IE, Chrome and Opera.

If you wish your code to be supported in all browsers you should use (in the first Hello World example) the following code.

```
function initialise()
{
  var mySpan = document.getElementById('jsOutput');
  if(mySpan.textContent != undefined) {
    mySpan.textContent = "Hello World";
  } else {
     mySpan.innerText = "Hello World";
  }
}
window.onload=initialise;
```

This code simply checks to see if *textContent* is defined in the browser to decide whether *textContent* or *innerText* is used.

JavaScript Comments

Single line comments start with *//*. These can be on a new line or at the end of a line of code, as shown below.

```
// This will write a header:
mySpan.innerHTML = "<h1>This is a header</h1>"
```

You can also use the HTML comment *<!--* to comment out a single line but not multiple lines.

```
<!-- This will write a header:
mySpan.innerHTML = "<h1>This is a header</h1>"
```

Multi-line comments start with */** and end with **/* as shown below.

```
/*
The code below will write
a header:
*/
mySpan.innerHTML = "<h1>This is a header</h1>"
```

Basic Programming in JavaScript

Naming Variables

Variables are containers for values (e.g.: x = 5) or expressions (e.g.: x = y + z). Variables in JavaScript are case sensitive – '*value*' and '*Value*' are two different variables and must begin with a letter or the underscore character ('_').

Declaring and Assigning Variables

The following JavaScript shows the variables *x*, *y*, *z* and *carMake*.

```
var y=5;
var y;
carMake = "Volvo";
z=4;
x=y+z;
```

Line 1 declares the variable *y* using the *var* statement and assigns it the value 5. Re-declaring the variable in line 2 will not make *y* lose its value. If you were to output the value of *x* in line 5 then ‘9’ would be displayed.

Line 3 assigns a value to the undeclared *carMake* variable. In assigning the value the variable is automatically declared.

Note: You should always declare your variables with the *var* keyword. If you do not then your variable will automatically assume a global scope within your script. This could potentially lead to serious issues in your code that would prove difficult to debug.

Accessing a Variable Value

A variable can be used in the place of the value it refers to. To see this, make the following amendments to “HelloWorld.js” and run “JSIntroHelloWorld.html”. You should see “Hello World” in your browser window.

```
function initialise()
{
  var message = "Hello World";
  var mySpan = document.getElementById('jsOutput');
  mySpan.textContent = message;
}
<!-- This is a comment
window.onload=initialise;
```

Arrays

Declaring an Array

The following are all valid ways of declaring an array.

You can declare your array so that you can add elements to it dynamically

```
var carMakers = new Array();    //declaration
carMakers[0] = "Volvo";         //assigning values
carMakers[1] = "Ford";
carMakers[2] = "Vauxhall";
```

or a condensed array

```
var carMakers = new Array("Volvo","Ford","Vauxhall");
```

or a literal array

```
var carMakers = ["Volvo","Ford","Vauxhall"];
```

Array elements begin at index 0 so that, for example, *carMakers[0]* is the first element in the *carMakers* array.

Data Types in an Array

The code snippets above have shown several ways of assigning values to an array.

Please note that the data type of an array element will depend upon the value that has been assigned to it.

```
var randomValues = ["Staffs Uni",2,"Hello World"];
```

The line of code above would result in the first and third elements having data type *string* and the second element would have data type *number*.

Accessing and Modifying Array Elements

Accessing a value in an array is similar to accessing a value in a variable except that you specify the index of the array element that is to be accessed.

Create a file called “ArrayExample.js” and save it containing the following code.

```
function initialise()
{
  var carMakers = ["Volvo","Ford","Vauxhall"];
  var mySpan = document.getElementById('jsOutput');
  mySpan.textContent = carMakers[1];
}
window.onload=initialise;
```

Make the following amendment to “JSIntroHelloWorld.html” and save it as “JSIntroArrayExample.html”.

```
head>
  <title>Introduction to JavaScript</title>
  <script type="text/javascript" src="ArrayExample.js"></script>
</head>
```

Run your web page to see the result.

Changing the value of an element in an array is just as straightforward.

Make the following amendment to “ArrayExample.js” and run it again to see the results.

```
function initialise()
{
  var carMakers = ["Volvo","Ford","Vauxhall"];
  carMakers[1] = "Audi";
  var mySpan = document.getElementById('jsOutput');
  mySpan.textContent = carMakers[1];
}
window.onload=initialise;
```

The highlighted line of code modifies the value of the second element in the *carMakers* array to the value “*Audi*”. This results in the value “*Audi*” being output instead of the original value “*Ford*”.

Operators

Arithmetic Operators

Operator	**Description**	**Example**	**Result (y=5)**
`+`	Addition	`x=y+2`	`x=7`
`-`	Subtraction	`x=y-2`	`x=3`
`*`	Multiplication	`x=y*2`	`x=10`
`/`	Division	`x=y/2`	`x=2.5`
`%`	Modulus (division remainder)	`x=y%2`	`x=1`
`++`	Increment	`x=++y`	`x=6`

`--`	Decrement	`x=--y`	`x=4`

Assignment Operators

Values in JavaScript are assigned using assignment operators. Given that *x=10* and *y=5*.

Operator	Example	Same As	Result
`=`	`x=y`		`x=5`
`+=`	`x+=y`	`x=x+y`	`x=15`
`-=`	`x-=y`	`x=x-y`	`x=5`
`*=`	`x*=y`	`x=x*y`	`x=50`
`/=`	`x/=y`	`x=x/y`	`x=2`
`%=`	`x%=y`	`x=x%y`	`x=0`

String Concatenation

The + operator can also be used to concatenate strings. When a string and a numerical value are concatenated the result will always be a string. This is shown in the following table.

Example	Result
`x="5"+"5"`	`x="55"`
`x="5"+5`	`x="55"`
`x=5+"5"`	`x="55"`
`x="Fred"+5`	`x="Fred5"`

Comparison Operators

Operator	Description	Example (x=5)
`==`	is equal to	`x==8` is false
`===`	is exactly equal to (value and type)	`x===5` is true `x==="5"` is false
`!=`	is not equal	`x!=8` is true
`>`	is greater than	`x>8` is false
`<`	is less than	`x<8` is true
`>=`	is greater than or equal to	`x>=8` is false
`<=`	is less than or equal to	`x<=8` is true

Comparison operators can be used in conditional statements to compare values and take action depending on the result. You will learn more about the use of conditional statements later in this tutorial.

JavaScript also contains a conditional operator that assigns a value to a variable based on some condition. This has the following syntax *variablename=(condition)?value1:value2*

For example

```
greeting=(visitor=="PRES")?"Dear Mr. President ":"Dear ";
```

If the variable visitor has the value of "PRES", then the variable greeting will be assigned the value "Dear Mr. President " else it will be assigned "Dear".

Conditional Statements

JavaScript has the following conditional statements:

- *if* statement - use this statement if you want to execute a block of code only if a specified condition is *true*
- *if...else* statement - use this statement if you want to execute a block of code if the condition is *true* and another block of code if the condition is *false*
- *if...else if....else* statement - use this statement if you want to select one of many blocks of code to be executed depending on multiple mutually exclusive conditions being *true* or another block of code if none of the conditions are *true*
- *switch* statement - use this statement if you want to select one of many blocks of code to be executed depending on multiple mutually exclusive conditions being *true* or another block of code if none of the conditions are *true*.

For the following examples save “JSIntroHelloWorld.html” as “JSIntroConditions.html” with the *src* attribute of the <*script*> element amended to “Conditions.js”.

Create the file “Conditions.js” based upon “HelloWorld.js” but with an empty *initialise()* function. You will be placing different code in the *initialise()* function in each of the examples below.

```
function initialise()
{
//Enter code here when indicated
}
window.onload=initialise;
```

The if Statement

The *if* statement will cause some code to be executed depending upon whether a specified condition is true.

```
if (condition) {
  code to be executed if condition is true
}
```

Amend “Conditions.js” to match the following then save it. In a browser view the web page “JSIntroConditions.html”.

```
function initialise()
{
//Comment on the price of coffee if it is over £2
  var coffeePrice = 5;
  var comment;
  if (coffeePrice > 2) {
    comment = "£" + coffeePrice + " is expensive for coffee";
    var mySpan = document.getElementById('jsOutput');
    mySpan.textContent = comment;
  }
}
window.onload=initialise;
```

The if...else Statement

If you want to execute some code if a condition is true and another code if the condition is not true, use the *if...else* statement.

```
if (condition) {
  code to be executed if condition is true
} else {
  code to be executed if condition is not true
}
```

The following code is an example of the *if...else* statement.

```
function initialise()
{
//Comment on the price of coffee
  var coffeePrice = 1;
  var comment;
  if (coffeePrice > 2) {
    comment = "£" + coffeePrice + " is expensive for coffee";
  } else {
    comment = "£" + coffeePrice + " is probably worth paying";
  }
  var mySpan = document.getElementById('jsOutput');
  mySpan.textContent = comment;
}
window.onload=initialise;
```

The if...else if...else Statement

The *if...else if...else* allows the execution of code dependent upon which of a number of conditions is true.

```
if (condition1) {
  code to be executed if condition1 is true
} else if (condition2) {
  code to be executed if condition2 is true
} else {
  code to be executed if condition1 and
  condition2 are not true
}
```

The following code is an example of an *if...else if...else* statement.

```
function initialise()
{
//Comment on the price of coffee
  var coffeePrice = 2;
  var comment;
  if (coffeePrice > 2) {
    comment = "£" + coffeePrice + " is expensive for coffee";
  } else if (coffeePrice <=2 && coffeePrice > 1) {
    comment = "£" + coffeePrice + " is probably worth paying";
  } else {
    comment = "£" + coffeePrice + " is definitely worth paying";
  }
  var mySpan = document.getElementById('jsOutput');
  mySpan.textContent = comment;
}
window.onload=initialise;
```

The switch Statement

You should use the *switch* statement if you want to select one of many blocks of code to be executed.

```
switch(n) {
  case 1:
    execute code block 1
    break;
  case 2:
    execute code block 2
    break;
  default:
    code to be executed if n is different from case 1 and 2
}
```

The expression *n* is compared with the values for each case in sequence. If there is a match, the block of code associated with that case is executed. The *break* statement is used to prevent the code from running into the next case automatically. Where there is no match, the *default* case is executed.

The following code is an example of a *switch* statement.

```
function initialise()
{
//You will receive a different greeting based
//on what day it is. Note that Sunday=0,
//Monday=1, Tuesday=2, etc.
  var comment;
  var d = new Date();
  theDay = d.getDay();
  switch (theDay) {
    case 1:
      comment = "Only Monday";
      break;
    case 2:
      comment = "Tuesday";
      break;
    case 3:
      comment = "Wednesday";
      break;
    case 4:
      comment = "Thursday";
      break;
    case 5:
      comment = "Finally Friday";
      break;
    case 6:
      comment = "Super Saturday";
      break;
    case 0:
      comment = "Sleepy Sunday";
      break;
    default:
      comment = "Don't know what day it is";
  }
  var mySpan = document.getElementById('jsOutput');
  mySpan.textContent = comment;
}
window.onload=initialise;
```

Loops

For the following examples create and save “JSIntroHelloWorld.html” as “JSIntroLoops.html” with the *src* attribute of the *<script>* element amended to “Loops.js”.

Create the file “Loops.js” based upon “HelloWorld.js” but with an empty *initialise()* function.

The for Loop

The *for* loop is used when you know in advance how many times the script should run.

```
for (n=startvalue; n<=endvalue; n++) {
  code to be executed
}
```

The example code below defines a loop that starts with *i=0*. The loop will continue to run as long as *i* is less than, or equal to *10*. *i* will increase by *1* each time the loop runs.

```
function initialise()
{
  var comment = "<ul>";
  for (var i=0; i<=10; i++) {
    comment += "<li>The number is " + i + "</li>";
  }
  comment += "</ul>";
  var mySpan = document.getElementById('jsOutput');
  mySpan.innerHTML = comment;
}
window.onload=initialise;
```

Note the use of the *innerHTML* property of *mySpan* because we want the HTML mark-up that we are including in our string to be interpreted as mark-up by the browser.

The while Loop

The *while* loop is used when you want the loop to continue executing while the specified condition is *true*.

```
while (n <= endvalue) {
  code to be executed
}
```

The example code below initialises the variable ‘*i*’ with the value *0*. The loop is then executed and will continue to run as long as *i* is less than, or equal to *10*. *i* will increase by *1* each time the loop runs.

```
function initialise()
{
  var i=0;
  var comment = "<ul>";
  while (i<=10) {
    comment += "<li>The number is " + i++ + "</li>";
  }
  comment += "</ul>";
  var mySpan = document.getElementById('jsOutput');
  mySpan.innerHTML = comment;
}
window.onload=initialise;
```

The do...while Loop

The *do...while* loop is a variant of the *while* loop. This loop will execute a block of code at least once, it will then repeat the loop as long as the specified condition is *true*.

```
do {
  code to be executed
} while (n<=endvalue);
```

The example code below initialises the variable '*i*' with the value *0*. The loop will then execute once. When the condition is evaluated to *false* the loop will exit.

```
function initialise()
{
  var i=0;
  var comment = "<ul>";
  do {
    comment += "<li>The number is " + i++ + "</li>";
  } while (i<0);
  comment += "</ul>";
  var mySpan = document.getElementById('jsOutput');
  mySpan.innerHTML = comment;
}
window.onload=initialise;
```

for...in Statement

The *for...in* statement is used to loop (iterate) through the elements of an array or through the properties of an object.

The code in the body of the *for...in* loop is executed once for each element/property.

```
for (variable in object) {
  code to be executed
}
```

The example code below declares an array and then uses the *for...in* loop to iterate through the array to output the element values using *i* as an index value.

```
function initialise()
{
  var comment = "<ul>";
  var carMakers = ["Volvo","Ford","Vauxhall"];
  for (var i in carMakers) {
    comment += "<li>" + carMakers[i] + "</li>";
  }
  comment += "</ul>";
  var mySpan = document.getElementById('jsOutput');
  mySpan.innerHTML = comment;
}
```

```
window.onload=initialise;
```

Functions

Functions associate a number of JavaScript statements so that they can be repeatedly called without having to be re-typed. Code within a function will not be executed unless the function is called.

Creating a Function

We have been using a function that we named “initialise” to contain code to run when the DOM was loaded in memory.

To create a function you use the following syntax

```
function functionName(optionalParameters) {
  some code
}
```

It is possible to have a function that accepts no parameters when called. This is true for the *initialise()* function. We can see in the code that it will accept no parameters because its parentheses (brackets immediately following the function name) are left empty. Following this declaration of the function are the function braces ‘*{*’ and ‘*}*’ that contain one or more JavaScript statements.

Function Examples

The following code example shows two functions that accept no parameters, a function that accepts one parameter, an anonymous function and the use of the return statement.

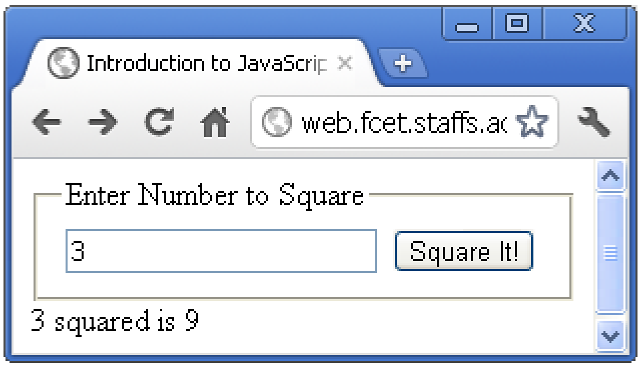
Create a file called “JSIntroSquareNumber.html” based upon “JSIntroHelloWorld.html”. Make the following amendments and save the file.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 1.0 Strict//EN"
"http://www.w3.org/TR/HTML1/DTD/HTML1-strict.dtd">
<html xmlns="http://www.w3.org/1999/HTML" xml:lang="en" lang="en">
<head>
  <title>Introduction to JavaScript</title>
  <script type="text/javascript" src="SquareNumber.js"></script>
</head>
<body>
  <div id="content">
    <form action="SquareNumberResult.php" method="POST" id="numberSquare">
      <fieldset>
        <legend>Enter Number to Square</legend>
        <input type="text" id="numToSquare" name="numToSquare" />
        <input type="submit" id="submitNum" name="submitNum" value="Square It!" />
      </fieldset>
    </form>
    <span id="jsOutput"></span>
  </div>
</body>
</html>
```

Create a file called “SquareNumber.js”. Type in the code below and save the file.

```
function squareNumber(numberToBeSquared)
{
  var result = numberToBeSquared * numberToBeSquared;
  return result;
}
function processNumber()
{
  var userValue = document.getElementById('numToSquare').value;
  if(userValue) {
    var strResult = userValue + " squared is " + squareNumber(userValue);
    document.getElementById('jsOutput').textContent = strResult;
  }
}
function initialise()
{
  var button = document.getElementById('submitNum');
  button.onclick = function() {
    processNumber();
    return false;
  }
}
window.onload=initialise;
```

View JSIntroSquareNumber.html in a browser and you should see a result similar to the following.



3 squared is 9

The result of the code is the output of the square of a number provided in a form by a user.

There are two extra functions introduced to the code, *processNumber()* and *squareNumber()*.

```
function squareNumber(numberToBeSquared)
{
  var result = numberToBeSquared * numberToBeSquared;
  return result;
}
function processNumber()
{
  var userValue = document.getElementById('numToSquare').value;
  if(userValue) {
    var strResult = userValue + " squared is " + squareNumber(userValue);
    document.getElementById('jsOutput').textContent = result;
  }
}
```

processNumber() is an example of a function that accepts no parameters and does not return a value to the statement that called it. Essentially *processNumber()* gets the user value to be squared, passes it to the function *squareNumber()* to perform the 'number squaring' operation and then outputs the result.

squareNumber() is an example of a function that accepts one parameter (*numberToBeSquared*) and returns a value. When the function is called a parameter is passed within parentheses. The call is made from the *processNumber()* function and can be seen highlighted in bold font below.

```
function processNumber()
{
  var userValue = document.getElementById('numToSquare').value;
  if(userValue) {
    var strResult = userValue + " squared is " + squareNumber(userValue);
    document.getElementById('jsOutput').textContent = result;
  }
}
```

The *return* Statement

This *squareNumber()* function also has a return statement.

```
function squareNumber(numberToBeSquared)
{
  var result = numberToBeSquared * numberToBeSquared;
  return result;
}
```

This returns a value to where the function was called. Effectively the function call takes on the value of the return statement. So if the value of *result* is *9* then the value *9* will be returned and concatenated into the string *strResult*.

Anonymous Functions

An anonymous function is a way of creating a function, containing one or more lines of code, that is executed as the script is executed. In the code below, the anonymous function is executed when the '*onclick*' event of the *button* object fires. The anonymous function follows the form *function() {...some code...}*.

```
function initialise()
{
  var button = document.getElementById('submitNum');
  button.onclick = function() {
    processNumber();
    return false;
  }
}
window.onload=initialise;
```

To explain this code in a little more detail. Some new code has been placed in the *initialise()* function to 'listen' for an '*onclick*' event on the element with *id="submitNum"*, which in this case is the submit button for the form. When the event fires, the code within the anonymous function will be executed. A call will be made to the *processNumber()* function and then a return value of *false* will be returned to the *onclick* event.

The result of this will be that the submission of the form (the submit button was clicked) will be cancelled because, in this case, JavaScript has handled the processing. If JavaScript was disabled then the form would be processed as normal with a 'POST' being made to the (currently fictitious) PHP file "SquareNumberResult.php". The PHP file would handle the processing as a fallback for the JavaScript not being able to. This is referred to as unobtrusive JavaScript.

The Lifetime of JavaScript Variables and Variable Scope

In the above function examples the variable '*numberToBeSquared*' is declared within the parentheses of the function as a newly created variable and it exists from the point at which the function is called until code execution is returned to the calling statement. In other words, the variable does not exist outside of the function. If the function is called again then '*numberToBeSquared*' will be created again.

The same is true for any variable declared within the function using the *var* keyword. It will only be accessible from within the function in which it is declared. This is referred to as a local variable. If the *var* keyword is omitted the variable will automatically become a global variable and will therefore be available outside the function. This is considered bad programming practice because it can lead to difficulties in identifying the global variables in a script.

The lifetime of a global variable begins when it is declared, and ends when the page is closed.

Create a new JavaScript file containing the following code.

```
var num1 = 5; //correctly declared global variable
function declareVariables() {
  var str = ""; //correctly declared local variable
  var num2 = 4; //correctly declared local variable
  num3 = 2; //poorly declared global variable
  str += "<p>num1: " + num1 + "</p>";
  str += "<p>num2: " + num3 + "</p>";
  str += "<p>num3: " + num2 + "</p>";
  return str;
}
function initialise()
{
  var str = ""; //correctly declared local variable
  str += declareVariables();
  str += "<p>num1: " + num1 + "</p>";
  str += "<p>num3: " + num3 + "</p>";
  str += "<p>num2:" + num2 + "</p>";
  document.getElementById('jsOutput').innerHTML = str;
}
window.onload=initialise;
```

The code above shows examples of variable scope. When you try to run the above code you will not get any output. You can get the code to work correctly if you comment out one of the lines. Can you figure out which one?