# The City Lit Institute

##### **Department of Computing**

## *Keeley Street, Covent Garden, London WC2B 4BA*

### **COURSE : Introduction to JavaScript**

### 

**LECTURER** : **ALEXANDER ADU- SARKODIE**

MSc. Telematics (IT & Telecom), MSc. Eng., Dip. Russ. Lang., Teach. Cert, AMIAEng (UK), MBCS (UK), MIfL(UK)

**LinkedIn -** <https://www.linkedin.com/in/alexander-adu-sarkodie-063b4b4a/>

**GitHub -** <https://github.com/kukuu?tab=repositories>

**Blog -** <http://www.goldhawk-college.com/istudio/tech-delivery.html>

**Email: [aadusarkodie1@gmail.com](mailto:aadusarkodie1@gmail.com)**

xi) Date() Object

The Date object is used to work with dates and times. Date objects are created with the Date() constructor.

All dates are calculated in milliseconds from 01 January, 1970 00:00:

There are four ways of initiating a date:

new Date() // current date and time  
new Date(milliseconds) //milliseconds since 1970/01/01  
new Date(dateString)  
new Date(year, month, day, hours, minutes, seconds, milliseconds)

Most parameters above are optional. Not specifying, causes 0 to be passed in.

Once a Date object is created, a number of methods allow you to operate on it. Most methods allow you to get and set the year, month, day, hour, minute, second, and milliseconds of the object, using either local time or UTC (universal, or GMT) time.

**Set Date**

We can easily manipulate the date by using the methods available for the Date object. In the example below we set a Date object to a specific date (14th January 2010):

Ex. 12

var myDate=new Date();  
myDate.setFullYear(2010,0,14);

And in the following example we set a Date object to be 5 days into the future:

Ex 13

var myDate=new Date();  
myDate.setDate(myDate.getDate()+5);

Ex3

**Compare two dates**

The Date object is also used to compare two dates. The following example compares today's date with the 14th January 2100:

Ex 14

var x=new Date();  
x.setFullYear(2100,0,14);  
var today = new Date();  
if (x>today){  
 alert("Today is before 14th January 2100. Sales hasn’t started yet");

//  
}  
else{  
alert("Today is after 14th January 2100. Sales begins at 10am");  
}

**5. The Array Object**

The Array object is used to store **multiple values** in a **single variable**. The object has **predefined properties** and **methods:**

**The Array Object Properties**

|  |  |
| --- | --- |
| **Property** | **Description** |
| Constructor | Returns the function that created the Array object's prototype |
| Length | Sets or returns the number of elements in an array |
| Prototype | Allows you to add properties and methods to an Array object |

**The Array Object Methods**

|  |  |
| --- | --- |
| **Method** | **Description** |
| concat() | Joins two or more arrays, and returns a copy of the joined arrays |
| indexOf() | Search the array for an element and returns it's position |
| join() | Joins all elements of an array into a string |
| lastIndexOf() | Search the array for an element, starting at the end, and returns it's position |
| pop() | Removes the last element of an array, and returns that element |
| push() | Adds new elements to the end of an array, and returns the new length |
| reverse() | Reverses the order of the elements in an array |
| shift() | Removes the first element of an array, and returns that element |
| slice() | Selects a part of an array, and returns the new array |
| sort() | Sorts the elements of an array |
| splice() | Adds/Removes elements from an array |
| toString() | Converts an array to a string, and returns the result |
| unshift() | Adds new elements to the beginning of an array, and returns the new length |
| valueOf() | Returns the primitive value of an array |

**Creating an Array**

An array can be created in three ways.

**Regular Array**

var myCars=new Array();   
myCars[0]="Saab";   
myCars[1]="Volvo";  
myCars[2]="BMW";

**Condensed Array**

var myCars=new Array("Saab","Volvo","BMW");

**Literal Array**

var myCars=["Saab","Volvo","BMW"];

**Access an Array.**

You refer to an element in an array by referring to the **index** number.

This statement access the value of the first element in myCars. You assign the returned value to a new variable.

var name=myCars[0];

And the following statement modifies the first element in myCars:

myCars[0]="Opel";

You can have different objects in an array. You can have functions in an Array. You can have arrays in an Array (Multi-dimensional Array):

myArray[0]=Date.now;  
myArray[1]=myFunction();  
myArray[2]=myCars;

**Array methods and properties**

Examples;

var x=myCars.length // the number of elements in myCars;  
var y=myCars.indexOf("Volvo") // the index position of “Volvo"

6. Functions

A function is a named sub-routine.It is a **piece of JavaScript code that can be** **executed once or many times** by the JavaScript application. **It is a way of packaging your JavaScript commands so you can easily reuse them every time you need the same piece of functionality implemented in your application**.

A function is created by using the JavaScript  *keyword* **“function”** followed by a given **identifier name**. The name must be **unique** ***through out the whole function***. The name is followed by a *pair* of **open plain brackets**, then a pair **of curly brackets** to contain the code to be executed. The function can be **called anywhere within the document** to execute **the statement** that it contains.

In this lesson, you will learn:

* what **functions** are and how they are used in JavaScript;
* how to **retrieve information from functions**;
* how to **give information to functions**;
* what **variable scope** is all about.

**This suggests that there are 2 distinct phases to a function:**

* The phase where thefunction is declared (created)**;**
* The phase where thefunction is called (used)**.**

There are a number of ways of declaring function types. Two most common ones are;

var artist = function () { //**anonymous function**

do something;

}

and

function getName() { // named function

do something;

}

Ex 15

<script type= “text/javascsript” language= “javascript”>

<!--

function call\_alert( ) {

alert( “my first javascript function”);

}

//-->

</script>

The **onload** attribute **(this is an event handler).** We canuseevent handlers to trigger the function when the document **is loaded**.

*Note the script is placed* ***in the header*** *in this case.*

<body **onload= “call\_alert( )”>**

<body onload = “call\_alert( )”>

Or unobtrusively as;

**window.onload = call\_alert;**

|  |
| --- |
| Note the absence of the parenthesis in this case.  Ex 16    <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8" />  <title> Package your JavaScript Code with Functions</title>  </head>  <body>  <h1> Package your JavaScript Code with Functions</h1>    <div>    <h2>Today's date is:</h2>    <span id="calendar"></span>    <input type="button" id="myButton" value="Get Date" onclick="showDate();" />    </div>    <script type="text/javascript">  function showDate() {    var today = new Date();  var myCalendar = document.getElementById("calendar");  var myButton = document.getElementById("myButton");  myCalendar.innerHTML = today.toDateString();  myButton.value = "Well done!";  }  </script>  </body>  </html> |
|  |

Ex 17: Word counter:

<form name="wordcount">

<textarea rows="12" name="wordcount2" cols="38" wrap="virtual"></textarea>

<br>

<input type="button" value="Calculate Words" onClick="countit()"> <input type="text" name="wordcount3" size="20">

</form>

<script type="text/javascript">

function countit(){

var formcontent = document.wordcount.wordcount2.value;

var content = formcontent.split(" ");

document.wordcount.wordcount3.value = content.length;

}

</script>

**6.2 Function Argument (Abstraction)**

The **plain brackets** that follow the name of all functions **may** be used to **contain data for use in the code to be executed**. This data is called the function “**argument**”. In the example that follows below the function call passes a string to the argument named “str” in the call\_alert( ) function for use in the code to be executed.

***You can use or reference this same block of code in different pages by assigning different string values. The function becomes universal, and can easily be propagated to different page environments.* *Re-using code (DRY – Don’t Repeat Yourself) and enhancing re-usability. This allows maintenance from a single source.***

function call\_alert(str ) {

alert( str);

}

}

<body onload = “call\_alert (‘passed value’)”>

It is important to note that string contained **inside the call** is enclosed in **single** quotes to differentiate it from the double quotes used to contain the **entire call**. Improper use will cause error. Assuming double quotes were used all over then there will be ***early termination*** after the second double quote and an error message will be produced.

Ex 18

<!DOCTYPE html>

<html>

<head>

<title>toString</title>

<script>

function myFunction(){

var fruits = ["Banana", "Orange", "Apple", "Mango"];

fruits.toString();

var x=document.getElementById("demo");

x.innerHTML=fruits;

}

window.onload = myFunction; //function call.

</script>

</head>

<body>

<p>The toString() method converts an array into a String and returns the result.

<br />

Note: The returned string will separate the elements in the array with commas.

</p>

<p id="demo">Click the button to convert the array into a String.</p>

</body>

</html>

Ex 19

<!DOCTYPE html>

<html>

<head>

<title>Push</title>

<script>

var fruits = ["Banana", "Orange", "Apple", "Mango"];

function myFunction(){

fruits.push("Kiwi")

var x=document.getElementById("demo");

x.innerHTML=fruits;

}

window.onload = myFunction;

</script>

</head>

<body>

<p>The push() method adds new items to the end of an array, and returns the new length.

</p>

<p id="demo">Click the button to add a new element to the array.</p>

<!-- <button onclick="myFunction()">Try it</button> -->

</body>

</html>

Ex 20

<!DOCTYPE html>

<html>

<head>

<title>Pop</title>

<script>

var fruits = ["Banana", "Orange", "Apple", "Mango"];

function myFunction(){

fruits.pop();

var x=document.getElementById("demo");

x.innerHTML=fruits;

}

</script>

</head>

<body>

<p>Removes the last element of an array, and returns the array</p>

<p id="demo">Click the button to remove the last array element.</p>

<button onclick="myFunction()">Try it</button>

</body>

</html>

**6.2 Writing External JavaScript Files**

Below is a simple external JavaScript document created by moving the script from the example above into a new separate document. This document is saved with a **.js** extension and should not have any html tags. Where they occur, they must be escaped appropriately so the script does not break the web document. Use known debugging methods to rectify any problems that persists. This document should neither have the **script** **tag** nor any **hiding syntaxes**. **Commenting** is allowed.

Using external js files allows for easier maintenance from a single source. Code can be reused and easily portable. Saving time to re-write the code into any page that may require similar routines (DRY – **D**on’t **R**epeat **Y**ourself).

1. *mylibrary.js:*

//begin

function call\_me (str){

alert(str);

}

//end

1. *Secondly embed a script tag in the calling page and point to the javascript file using the src attribute in the tag*. *As seen in the two examples below.*

Ex 21

<!DOCTYPE html>

<html>

<head>

<title>Using External JavaScript Files 1</title>

<script language="javascript" type="text/javascript" **src=" mylibrary.js">**</script>

</head>

<body onload="call\_me('This text is personalised to this page. It is using external js file to load')">

</body>

</html>

Ex 22

<!DOCTYPE html>

<html>

<head>

<title>Using External JavaScript Files</title>

<script language="javascript" type="text/javascript" src="mylibrary.js"></script>

</head>

<body onload="call\_me('This is a new second page. It is using the same external js file to load as the first. An example of code re-usability')">

</body>

</html>

**6.3 Nested functions and closures**

Variables declared **inside a function** are called **local variables** and can only be used **within that function**. Conversely, “**global**” variables are declared **outside functions and can be accessed by any function within the same document.**

You can nest a function within a function. The **nested** (inner) function is **private** to its containing (outer) function. It also forms a ***closure***.

**A closure is an expression (**typically a function**) that can have free variables together with an environment that binds those variables (that "closes" the expression).**

Since a nested function is a closure, this means that a nested function can "**inherit**" the arguments and variables of its containing function. In other words**, the inner function contains the scope of the outer function.**

To summarize:

* The inner function can be accessed only from statements in the outer function.
* The inner function forms a closure: the inner function can use the arguments and variables of the outer function, while the outer function cannot use the arguments and variables of the inner function.

Ex 23

<!DOCTYPE html>

<html>

<head>

<title>multipleFunction:Building Calculator Function</title>

<script language="javascript" type="text/javascript">

function addSquares(a,b) {

function square(x) {

return x \* x;

}

return square(a) + square(b);

}

//a = addSquares(2,3); // returns 13

//b = addSquares(3,4); // returns 25

//c = addSquares(4,5); // returns 41

document.write(addSquares(2,3));

` </script>

</head>

<body onload="addSquares(2,3)">

</body>

</html>

Ex 24

<script>

function sayHello2(name) { //function  
   var text = 'Hello ' + name; // statement  
   var sayAlert = function() { //publish

alert(text);

}  
  return sayAlert;

}

</script>

Ex 25

<script>

function buildList(list) {  
   var result = [];  
   for (var i = 0; i < list.length; i++) {  
     var item = 'item' + list[i];  
     result.push( function() {

alert(item + ' ' + list[i])

} );

}  
   return result;

</script>

6.4 Making Multiple Arguments

JavaScript functions may contain **multiple arguments if they are separated by a *comma***. Note the **number of arguments** defined when the function **is declared** must be **exactly matched** by the **number of arguments contained in any call to that function***. Any unused arguments in assigning data must be replaced by* ***empty string*** *or space. This is achieved by using a pair of quotes* acting as white space**.** In the example below three arguments have been declared in the function, so **any call** to it **must contain three argument values:**

***Note:***

**1. You can use this same block of code in different pages of your site by passing different string values to the onload attribute of the body. Mind very carefully that, unused arguments in the call to the function must be replaced by an empty string denoted by** *‘’* **or there will be error messages showing up. The called argument in the function call MUST be separated by commas and each value held in a delimiter.**

Ex 26

var a, b, c ; //global variables

function call\_alert ( str1, str2, str3 ) {

//these three arguments are assigned to the global variables.

a = str1;

b = str2;

c = str3;

alert ( a + b + c); //print the values (variables) stored in the argument

}

<body onload = “call\_alert (‘Great’, ‘ ’,’Javascript‘)”>

The caller passes **three strings** to the function with the value of the **second merely a space**. The function assigns the argument values to the three global variables, then calls alert ( ) to display the global values as **a single concatenated string.**

* 1. **Unobtrusive JavaScript, and the Document Object Model**

**Document Object model** (DOM) provides:

* An **object model**, and alongside it
* an **API** (Application Programme Interface)

for a document. One of the best ways to visualize the DOM for a webpage is to use the DOM Inspector that comes with Mozilla or Chrome browsers.

Within the document object model all page elements are placed in a **tree like hierarchy**.

When you are working with the DOM, the ideal way to later **access** an element on the page is by giving it a unique **identifier**, or **ID**. For a group of elements you can use a **className** or common **attribute**s. Then, you can use the **DOM method** below to access elements and properties of a node:

*document.getElementById().*

This method then **accesses** the given element and enables you to go on from there: **Modify** the element, **append** sub-elements or otherwise **navigate** through the DOM tree.

The DOM API support methods to not only access elements in the DOM tree, but also **add** and **remove** elements. Therefore it is possible to modify virtually anything on the page.

The JavaScript you may know has already been separated from the mark-up to a certain degree. The functions that do all the work are contained in an external file. The **problem** lies with the **in-line handlers**.

Using an attribute like ***onclick*** in the mark-up is just as inefficient as using the ***style*** attribute in CSS as an in-line directive (as some of you may know). It would be much better if we could use a hook, like ***class***or ***id***, **to tether the behaviour to the mark-up without intermingling it**. This is how the mark-up could indicate that a links in a page with class of assigned value **popup** should have the popUp or overlay function executed when it is called to action (**CTA**):

<a href= “example.html” class= “popup”>example</a>

This is entirely possible. Events do not have to be handled in the mark-up. **You can attach an event to an element in an external JavaScript file**. The tricky part is figuring out **which** **element** should have the event attached.

If you want to attach an event to an element with unique *id*, you can simply use *getElementById;* i.e

***getElementById(id).event= action;***

With multiple elements, you can use a combination of ***getElementsByTagName* , *getAttribute*  and *getElementsByClassName*** to attach events to elementswith specific attributes.

Below is the procedure, for opening up a series of pop-up links from within a page with common class name **popup**.

i) Make an **array** of all the links in the document

ii) **Loop** through this array

iii) If a link has the **class "popup**", execute this **behaviour** when the link is **clicked**

iv)Pass the **value** of the links href attribute to the **popUp** function.

v) **Cancel** the **default** behaviour so that the link isn't **followed in the original window**.

And this is how it should look like;

var links= document.getElementsByTagName("a"); //make an array of link

for (var i=0; i < links.length;i++) { //loop through

if(links[i].getAttribute("class")=="popup"){ //Note use of == operator

links[i].onclick=function() {

openUp(this.getAttribute("href"));

return false; //Cancel default behaviour

}

}

}

Now the **connection** between the **links** and the **behaviour** that should occur when the links are clicked has been **moved out** of the mark-up and into the external JavaScript file. This is **unobtrusive JavaScript**.

As example again, rather than adding the onchange attribute explicitly, the relevant element(s) are simply identified, for example by class, id or some other means in the markup:

<input type="text" name="date" class=”sname” id="date" />

A script that runs when the page is first loaded into the browser can then look for the relevant element(s) and set them up accordingly:

Next fine tune the code using wrapper functions to load into browser on on-load. Note the **object model** is a property of the **window object** and so it will inherit the windows **on-load event handler**

**The unobtrusive solution is to register the necessary event handlers programmatically, rather than inline.**

Finally, don’t forget to check for **graceful degradation**. Add some code to test for DOM.

Ex 27

<!DOCTYPE html>

<html lang="en-gb">

<head>

<title>oJS 5</title>

<script language=”javascript” type="text/javascript" src="popstuff.js"></script>

</head>

<body>

<a href="poppage1.html" class ="popup">popup page1</span></a>

<a href="poppage2.html" class ="popup">popup page2</span></a>

<a href="poppage3.html" class="popup">popup page3/span></a>

</body>

</html>

**popstuff.js**

window.onload = prepareLinks; //referencing the function.

function prepareLinks() {

var links= document.getElementsByTagName("a");

for( var i=0; i<links.length;i++) {

if(links[i].getAttribute("class")=="popup"){

links[i].onclick=function() {

openUp(this.getAttribute("href"));

return false;

}

}

}

}

function openUp(winURL) {

window.open(winURL,"popup","width=320, height=480");

}

**7.1 DOM element properties**

### Element properties

**Properties Description**

**attributes[]** Returns an array containing all the attributes defined for the element in question, including custom attributes. IE6 returns not just attributes explicitly defined, but those of the element's internal DTD as well. In Firefox, attributes[] work more as expected, returning only user defined attributes, and even reflect changes done by scripting to an attribute. Each attribute[] element returned supports a **name** and **value** property to retrieve additional information about the attribute.

Ex 28

var imageattributes=document.getElementById("myimage").attributes;

imageattributes[0].name //name of the first attribute of "myimage"

imageattributes[0].value //value of the first attribute of "myimage"

imageattributes.**getNamedItem**("src").value //value of the "src" property of "myimage"

eg. <img src= “antelope.gif” alt=”The antelope” id=”ant” />

**childNodes[]** Returns an array of all of the child nodes of an element as **objects**. Use the properties "**nodeName**" and "**nodeType**" to retrieve additional information about a node.

Ex 29

//access some <ul> element  
var mylist=document.getElementById("mylist")  
for (i=0; i<mylist.childNodes.length; i++) {  
 if (mylist.childNodes[i].nodeName=="LI") {  
 //do something

}  
}

**Html**

<div>

<h2>Hey</h2>

<p>What you doing?!</p>

<ul id= “myList”>

<li>1</li>

<li>2</li>

<li>3</li>

</ul>

</div

**className** Returns the CSS class attribute of an element.

Ex 30

document.getElementById("test").className="class1" //Assign the class "class1" to element  
document.getElementById("test").className +=" class2" //Assign an additional "class2" class to element

**clientWidth** A **cross browser** property that returns the **viewable** width

of the content on the page, not including **borders**,

**margins**, or **scrollbars** (overflowing content).

Ex 31

var pagewidth=document.body.clientWidth

**clientHeight** A **cross browser** property that returns the viewable height

of the content on the page, not including borders,

margins, or scrollbars (overflowing content).

**innerHTML** A **cross browser** (non W3C DOM) property that lets you

easily change the HTML contents of an element. Generally, this property can only be invoked **after** the document has **fully loaded** (Synchronous processes). Quite useful in synchronous processes.

Ex 32

<p><b>Old paragraph text</b></p>  
<script type="text/javascript">  
 window.onload=function() {  
 document.getElementsByTagName("p")[0].innerHTML="<b>new paragraph text</b>";  
 }  
</script>

**firstChild** Returns a reference to the first child of an element (below).

**nodeName** Returns a string indicating the name of the node, in

the case of elements, its tag name. Returned value is in uppercase.

Ex 33

if (document.getElementById("test").firstChild.nodeName=="DIV") {  
 alert("This is a DIV");

}

**nodeType** Returns an integer indicating the type of a node.

**Style** References the style object of an element,

in turn accessing and modifying individual style attributes' values.

Ex 34

document.getElementById("test").style.backgroundColor="yellow";

**nodeValue** Read/write property that reflects the value of a node. For text nodes, the content of the node is returned, while for attribute nodes, the attribute value. Null is returned for Document and element nodes.

Use this property to alter the contents of a text

or attribute node.

Ex 35

<body>  
<div id="test">Old text</div>  
  
<script type="text/javascript">  
 if (document.getElementById("test").firstChild.nodeName=="#text") {  
 document.getElementById("test").firstChild.nodeValue="New text";  
 }  
</script>

**7.2 DOM – Adding Elements**

Ex 36

<!DOCTYPE HTML>

<html>

<head>

<title>DOM 3 - Removing elements</title>

<script language="JavaScript" type="text/javascript">

initAddItem: function addItem() {

var list = document.getElementById("list");

var newNode = document.createElement("li");

var newTextNode = document.createTextNode("Hey, appended this text to the dynamically created list. Isn't this great! I love DOM");

newNode.appendChild(newTextNode);

list.appendChild(newNode);

list.insertBefore(newNode,list.firstChild);

}

window.onload=addItem;

</script>

</head>

<body>

<ul id="list"><li>item</li></ul>

</body>

</html>

**DOM - Changing complete HTML Fragments**

Ex 37

<!DOCTYPE HTML>

<html>

<head>

<title>innerHtml -changing complete HTML Fragments </title>

<script language="JavaScript" type="text/javascript">

var nr = 1;//this is a counter

function addItem() {

var list = document .getElementById("list");// get ID of the list item

nr++;

var newNode = "<li>item" + nr + "</li>";

list.innerHTML += newNode;

}

</script>

</head>

<body onload="addItem();">

<ul id="list">

<li>item 1</li>

</ul>

</body>

</html>

**Generate Dynamic Object (template)**

Ex 38

<!DOCTYPE HTML>

<html>

<head>

<title>Styled created table from JavaScript Data - 2</title>

<script language="JavaScript" type="text/javascript">

//Author : Alexander Adu-Sarkodie

//To be used with above instruction ONLY

function createTable(data) {

var table = document.createElement("table");

table.style.border = "1px solid #ffcc33";

var thead = document.createElement("thead");

thead.style.padding = "5px";

var tr = document.createElement("tr");

for (var i = 0; i < data[0].length; i++) {

var th = document.createElement("th");

th.style.border = "2px solid #ff0000";

var newText = document.createTextNode(data[0][i]);

th.appendChild(newText);

tr.appendChild(th);

}

thead.appendChild(tr);

table.appendChild(thead);

var tbody = document.createElement("tbody");

for (var i =1; i < data.length; i++){

var tr = document.createElement("tr");

for (var j = 0; j < data[i].length; j++) {

var td = document.createElement("td");

td.style.padding = "5px";

td.style.border = "2px solid #00ff00";

var newText = document.createTextNode(data[i][j]);

td.appendChild(newText);

tr.appendChild(td);

}

tbody.appendChild(tr);

}

table.appendChild(tbody);

return table;

}

window.onload = function() {

var table = createTable ([

["1","2","3","4"],

["One","Two","Three","Four"],

["Un","Deux","Trois","Quatre"],

["eins","zwet","dret","vier"]

]);

document.body.appendChild(table)

}

</script>

</head>

<body>

</body>

</html>

#### Event Delegation and Triggers

JavaScript **events** are the **bedrock** of all interactivity on web page .

In **traditional** event handling you **add** or **remove** event handlers from each element

as needed.

However, event handlers can potentially cause **memory leaks** and **performance**

**degradation** - the more you have, the greater the risk. JavaScript event delegation is a simple

technique by which you add a **single** event handler to a **parent element** in order to

avoid having to **add event handlers to multiple child elements**.

## How it works

Event delegation makes use of two features of JavaScript events: event **bubbling** and **target** element . **When an event is triggered on an element, for example a mouse click on a button, the same event is also triggered on all of that element’s ancestors**. This process is known as event **bubbling**; the event **bubbles up from the originating element to the top of the DOM tree.**

The **target** element of any event is the **originating element**, the button in our example, and is stored in a **property** of the **event object**.

**Using event delegation it’s possible to add an event handler to an element, wait for an event to bubble up from a child element and easily determine from which element the event originated.**

**In short, bubbling is the notion that for instance a clicked element registers a click, then the event "echoes" up through the nodes in the DOM to the top level -- however, you can grab that event and determine the originating source or "target" object on the page.**

Ex 39

In the example below, when the event bubbles up to the UL element, you check the event object's target property to gain a reference to the actual clicked node.  Here's a very basic JavaScript snippet which illustrates event delegation:

// Get the element, add a click listener...

document.getElementById("parent-list").addEventListener("click",function(e) {

// e.target is the clicked element!

// If it was a list item

if(e.target && e.target.nodeName == "LI") {

// List item found! Output the ID!

console.log("List item ",e.target.id.replace("post-")," was clicked!");

}

});

Ex 41

(To be completed in class).

Now let's have a parent DIV with many children but all we care about is an A tag with the "classA" CSS class :

// Get the parent DIV, add click listener...

document.getElementById("myDiv").addEventListener("click",function(e) {

// e.target was the clicked element

if(e.target && e.target.nodeName == "A") {

// Get the CSS classes

var classes = e.target.className.split(" ");

// Search for the CSS class!

if(classes) {

// For every CSS class the element has...

for(var x = 0; x < classes.length; x++) {

// If it has the CSS class we want...

if(classes[x] == "classA") {

// Bingo!

console.log("Anchor element clicked!");

// Now do something here....

}

}

}

}

});

Event delegation is also a great way to avoid **crippling** the user's browser when you're working with a huge document. As another example , if you have a table with thousands of cells, and you want something to happen when the user clicks on one, you won't want to attach a click handler to every single one of them . Instead, you can attach the click handler to **a single table element** and use **event.target** to **pinpoint the cell that is being clicked**. Below is how a jQuery program will handle such an event.

### **Conclusions about Event Delegation**

* Is easier to assign
* Can consolidate all events into a nicer centralized package which distributes functionality from one set of events
* Persists after the DOM has loaded, and if it is modified
* Uses less memory footprint in your browser window
* Consequently, may perform better
* On a large scale desktop-like app, it *will* perform better

#### The Window Object, Statements & Loops

The window object represents a window containing a DOM document; the document property points to the DOM document loaded in that window.

**Statements** are used in JavaScript to **progress the execution** of the JavaScript application. They may define **loops within the code or be simple terms to evaluate**.

We will be covering conditional testing and the different types of loops.

9.1 While Loops

In this example we will make sure that the **user enters his or her name at all cost.** If the evaluation **returns true then the code in the statement block will be executed**. After the code has **been executed** the test expression will **again be evaluated and the loop will continue until the loop returns false.**

Ex 42

<!DOCTYPE html>

<html>

<head>

<title>code checking name entry</title>

</head>

<body>

<script language=”javascript” type= “text/javascript”>

var myname = " ";

myname = **prompt** (" Enter your name here, please. " **,** myname )

**while** (myname = = “” || myname = = "undefined" ) {

myname = **prompt** ( "Please try again typing your name." )

}

**document . write** ("Hello " + myname)

</script>

</body>

</html>

Note : We need to look at the use of the variable **myname.** First we give it null value and then call this *initialising* the variable :

**myname = " "**

Then we use the **variable within the prompt method to receive the user's input into the variable**

myname = **prompt** (" Enter your name here, please. " **,** myname )

Note that if the user does not enter a name, our variable **myname** will have the value *null*. If that happens we have to ask the user again to enter a name . We can do this by using the **while** loop

**while** (myname = = “” || myname = = "undefined" ) {

myname = **prompt** ( "Please try again typing your name." )

}

Note once again the **while** keyword which is followed by a condition (inside the parenthesis) which controls the loop. As long as the condition is true the loop will keep going around executing the statements that are included within { }. In this example the loop will keep asking the user for a name as long as he/she does not enter a name.

Ex 43

<!DOCTYPE html>

<html>

<head>

<title>code checking name entry</title>

</head>

<body>

<script language="javascript" type="text/javascript">

var firstName = '';

var lastName = '';

firstName = prompt('Enter your first name',firstName);

while (firstName == ‘’ || firstName == 'undefined') {

firstName = prompt('Try again entering your first name');

}

if (firstName != ' ' && firstName != 'undefined') {

lastName = prompt('Enter your last name',lastName);//this is a more rigid test

while (lastName == ‘’ || lastName == 'undefined') {

lastName = prompt('Try again entering your last name');

}

}

//finally out put data to browser

document.write('Hello ' + firstName + ' ' + lastName);

</script>

</body>

</html>

Ex 44

<!DOCTYPE html>

<html>

<head>

<title>JavaScript Calendar</title>

<script type="text/javascript">

var month\_array = new Array();

month\_array[0] = "January";

month\_array[1] = "February";

month\_array[2] = "March";

month\_array[3] = "April";

month\_array[4] = "May";

month\_array[5] = "June";

month\_array[6] = "July";

month\_array[7] = "August";

month\_array[8] = "September";

month\_array[9] = "October";

month\_array[10] = "November";

month\_array[11] = "December";

document.write('<select name="day">');

var i = 1;

while ( i <= 31 ) {

document.write('<option value=' + i + '>' + i + '</option>');

i++;

}

document.write('</select>');

document.write('<select name="month">');

var i = 0;

while ( i <= 11 ) {

document.write('<option value=' + i + '>' + month\_array[i] + '</option>');

i++;

}

document.write('</select>');

document.write('<select name="year">');

var i = 1900;

while ( i <= 2005 ) {

document.write('<option value=' + i + '>' + i + '</option>');

i++;

}

document.write('</select>');

</script>

</head>

<body>

</body>

</html>

Ex 45

<!DOCTYPE html>

<html>

<head>

<title>Break</title>

<script type="text/javascript">

document.write("<p><b>Example of using the break statement:</b></p>");

var i = 0;

for (i=0; i<=10; i++) {

if (i==3){break;} //The counter will stop at 3

document.write("The number is " + i);

document.write("<br />");

}

</script>

</head>

<body>

</body>

</html>

Ex 46

<!DOCTYPE html>

<html>

<head>

<title>Continue</title>

<script type="text/javascript">

document.write("<p><b>Example of using the continue statement:</b><p>");

var i = 0;

for (i=0; i<=10; i++) {

if (i==3){continue} // The counter will skip 3

document.write("The number is " + i);

document.write("<br />")

}

</script>

</head>

<body>

</body>

</html>

* 1. **Properties and methods of the window object**

The properties and methods of the window.document object enable a web page document to be changed at runtime.

Using JavaScript to **assign new values to properties of the window.document object causes the** **web browser to update the appearance of the web page in line with the new values**. This can be simply illustrated by **assigning new values** to the **document.bgColor** and **document.fgColor** properties to change the **background and foreground colours of a page**.

In the example below, the setcolour( ) function runs when the **user pushes the button and the page appearance is changed**:

Ex 47

<!DOCTYPE html>

<html>

<head>

<title>window properties</title>

<script language=”javascript” type="text/javascript">

function poll(){

for (var i=0; i<document.f.rad1.length; i++){

if (document.f.rad1[i].checked){

document.bgColor=document.f.rad1[i].value;

//OR fgColor

}

}

}

</script>

</head>

<body>

<div align="center"><h2>Changing background colour</h2></div>

<div align="center">

<form name="f">

<input type="radio" name="rad1" value="white">white

<input type="radio" name="rad1" value="silver">silver

<input type="radio" name="rad1" value="skyblue">skyblue

<input type="button" value="Change colours" onclick="poll()">

</form>

</div>

</body>

</html>

Ex 48

The For loop is executed till a specified condition returns false. It takes 3 arguments and looks as follows:

for (initialization; condition; increment/decrement) {

// statements

}

When the For loop executes, the following occurs:

The initializing expression is executed. This expression usually initializes **one** or **more** loop counters, but the syntax allows an expression of any degree of complexity.

The condition expression is evaluated. If the value of condition is true, the loop statements execute. If the value of condition is false, the FOR loop terminates.

The update expression increment or decrement executes. The statements execute, and control returns to step 2.

The following example generates a multiplication table 2 through 9. **Outer loop** is responsible for generating a list of ***dividends***, and **inner loop** will be responsible for generating lists of ***dividers*** for each individual number:

Ex 49

<!DOCTYPE html>

<html>

<head>

<title>Loops 1</title>

<script type="text/javascript">

document.write("<h1>Multiplication table</h1>");

document.write("<table border=2 width=50%");

for (var i = 1; i <= 9; i++ ) { //this is the outer loop

document.write("<tr>");

document.write("<td>" + i + "</td>");

for ( var j = 2; j <= 9; j++ ) { // inner loop

document.write("<td>" + i \* j + "</td>");

}

document.write("</tr>");

}

document.write("</table>");

</script>

</head>

<body>

</body>

</html>

Ex 50

<!DOCTYPE html>

<html>

<body>

<script>

var i;

var mycars = new Array();

mycars[0] = "Saab";

mycars[1] = "Volvo";

mycars[2] = "BMW";

for (i=0;i<mycars.length;i++) {

document.write(mycars[i] + "<br />");

}

</script>

</body>

</html>

Ex 51

<!DOCTYPE html>

<html>

<head>

<title>Loops 2</title>

<script type="text/javascript">

function howMany (selectItem) {

var numberSelected=0;//create a variable to store values. Set to 0

for (var i=0; i < selectItem.options.length; i++) {//**options** is DOM property of selected items

if (selectItem.options[i].selected == true)

numberSelected++;//add to the variable. Increasing the counter.

}

return numberSelected// Send or post value to function/application

}

</script>

</head>

<body>

<p>Next example creates a function containing the For statement that counts the number of selected options in a list. The For statement declares the variable i and initializes it to zero. It checks that i is less than the number of options in the Select object, performs the succeeding if statement, and increments i by one after each pass through the loop.</p>

<form name="selectForm">

<p>Choose some book types, then click the button below:</p>

<select multiple name="bookTypes" size="8">

<option selected> Classic </option>

<option> Information Books </option>

<option> Fantasy </option>

<option> Mystery </option>

<option> Poetry </option>

<option> Humor </option>

<option> Biography </option>

<option> Fiction </option>

</select>

<input type="button" value="How many are selected?" onclick="alert ('Number of options selected: ' + howMany(document.selectForm.bookTypes))">

</form>

</body>

</html>

* 1. **Conditional *if* statement**

The “**if**” keyword is used to perform the **basic conditional JavaScript test**, to evaluate an expression for a **Boolean value**. The *statemen*t **following** the evaluation will **only be executed** when the expression **returns true**. Below is the syntax for the “if” statement.

if (test expression ) {

statement to execute when true;

}

**11.2 Using The *if else* statement with *confirm***

The JavaScript **“else”** keyword can be used with an “**if**” statement to provide **alternative code to execute in the event that the test expression returns false**. This is known as **conditional branching** and has this syntax:

if (test expression ) {

do this ;

}

else {

do this ;

}

The JavaScript **else if** keyword is used to provide response to when there are a **number** of **options or tailored responses to be considered**  for evaluation by the application or program.

Ex 52

<!DOCTYPE html>

<html>

<head>

<title> using the else if </title>

<script language = "javascript” type=”text/javascript”>

**function** validate ( ){

**if**  ( document.myform**.**firstname**.**value = = " " ){

**alert** ( "You must enter your first name" ) **; //** checks for name entry

return **;**

}

**else if** ( document.myform.surname.value = = “ ” ){ //checks for surname entry

alert (“You must enter your surname.”) ;

return;

}

submitnnow = **confirm** ("Do you want to submit now ?"); //disclaimer or flag

**if** (submitnow){

document**.**myform**.**submit ( ) **;**

}

**else** {

return **;**

}

//document.getElementById(“send”).onclick = validate;

}

</script>

</head>

<body>

<form name ="myform">

<h1>booking form</h1>

first name : <input name = "firstname">

<br />

surname :<input name ="surname">

<input type = "button" id= “send” value = "book now" **onclick** = "validate ( )">

</form>

</body>

</html

The function we have written validates the name and surname. This forces the user to enter his name and surname in the form.

The function code is included in the HEAD section. This is always the case when we are building our own functions.

We call the function by using its name **validate().** This function is invoked or triggered by the event handler **onclick.** Used in the context here as an *attribute***.** In our case the validate function will be used to check the name, and surname.

**11.3 Email Validation**

Ex 53

This simple email validation script tests for the presence of the **‘@**’ character in an email address

<!DOCTYPE html>

<html>

<head>

<title>Email validation</title>

<script type="text/javascript">

function validation(){

if(document.f.email.value.indexOf("@") == -1) {

alert("Email address is absent or incorrect");

return false;

}

}

</script>

</head>

<body>

<div align="center">

<form name="f" method="post" action="mailto:abc@aol.com" enctype="text/plain" **onsubmit**="**return** validation()">

Enter your email address:<input type="text" name="email" value="">

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

</form>

</div>

</body>

</htm>

**A more robust validation script**

Ex 54

<!DOCTYPE html>

<html>

<head>

<title>email validation</title>

<script type="text/javascript" language="javascript">

<!--

function send\_if\_valid() {

if(document.f.email.value == "") {

fail("you must enter your email address");

}

else if(document.f.email.value.indexOf("@") == -1) {

fail("No '@' in the address");

}

else {

var addr = document.f.email.value.split("@");

if (addr[0].length < 1 ) {

fail("User address absent");

}

else if (addr[1].indexOf(".") == -1 ) {

fail("No dot");

}

else if (addr[1].length < 3) {

fail("Domain incorrect");

}

else {

document.f.submit();

}

}

}

function fail(msg) {

alert("Email Address Error : \n" + msg);

}

//-->

</script>

</head>

<body>

<form name="f" method="post" action="#">

<!-- <form name="f" method="post" action="mailto:bob@bob.com"> -->

<p>Please enter email address:

<input name="email" type="text" value="">&nbsp;<input type="button" value="submit form" onclick="send\_if\_valid()">

</p>

</form>

</body>

</html>

**Class Exercises**

**1. The Date() object**

<!doctype html>

<html lang="en">

<head>

<title>Using Date Object</title>

</head>

<body>

<h1>Current date and time</h1>

<script type="text/javascript" src="timegreet.js"></script>

</body>

</html>

**2. Working with String Arrays - pop() method**

<!DOCTYPE html>

<html>

<head>

<title>Pop</title>

<script>

var fruits = ["Banana", "Orange", "Apple", "Mango"];

function myFunction(){

fruits.pop();

var x=document.getElementById("demo");

x.innerHTML=fruits;

}

</script>

</head>

<body>

<p>Removes the last element of an array, and returns the array</p>

<p id="demo">Click the button to remove the last array element.</p>

<button onclick="myFunction()">Try it</button>

<p class="home"><a href="http://www.goldhawk-college.com/istudio/responsive-design/responsive-design-1.html">Home</a></p>

</body>

</html>

**3. Generate dynamic URL**

<!DOCTYPE HTML>

<html>

<head>

<title>Spliting strings:Split functions</title>

<script type="text/javascript">

var myString = "zero one two three four";

var mySplitResult = myString.split(" ");

for(i = 0; i < mySplitResult.length; i++){

document.write("<br /> This is image " + (i+1) + " = "+"<img src=" + mySplitResult[i]+".gif"+">");

}

</script>

</head>

<body>

<p><b>Class Exercise:</b>Using JavaScript, transform the string

"One two three four and five" as part of an application code

to display five images.</p>

</body>

</html

**4. Using string push()**

<!DOCTYPE html>

<html>

<head>

<title>Push</title>

<script>

var fruits = ["Banana", "Orange", "Apple", "Mango"];

function myFunction(){

fruits.push(["Kiwi","Guava","Pineapple"])

var x=document.getElementById("demo");

x.innerHTML=fruits;

}

window.onload = myFunction;

</script>

</head>

<body>

<p>The push() method adds new items to the end of an array, and returns the new length.

</p>

<p id="demo">Click the button to add a new element to the array.</p>

<!-- <button onclick="myFunction()">Try it</button> -->

<p class="home"><a href="http://www.goldhawk-college.com/istudio/responsive-design/responsive-design-1.html">Home</a></p>

</body>

</html>

**5. Using subString()**

<!DOCTYPE html>

<html>

<head>

<title>Using indexOf and substr to get user id and domain </title>

<script language="javascript" type="text/javascript">

var my\_email="1234mike@marksandspencer.com"

//var ind=my\_email.indexOf(".");

var my\_slice=my\_email.substr((9));

document.write(my\_slice);

</script>

</head>

<body>

<p>Using substr function to get userid and domain part. Here is the code </p>

</body>

</html>

**7. Event delegation, Unobtrusive JavaScript and DOM manipulation**

A)

<!DOCTYPE HTML>

<html lang="en-gb">

<head>

<title>oJS 5</title>

<script type="text/javascript" src="popstuff.js"></script>

</head>

<body>

<a href="poppage1.html" class="popup">popup page1</a>

<br />

<a href="poppage2.html" class="popup">popup page2</a>

<br />

<a href="poppage3.html" class="popup">popup page3</a>

</body>

</html>

B) popstuff.js

window.onload = prepareLinks;

function prepareLinks() {

var links= document.getElementsByTagName("a")

for( var i=0; i<links.length;i++) {

if(links[i].getAttribute("class")=="popup"){

links[i].onclick=function() {

popUp(this.getAttribute("href")); return false;

}

}

}

}

function popUp(winURL) {

window.open(winURL,"popup","width=320, height=480");

}

**8. Using Conditions and For Loop**

<!doctype html>

<html>

<head>

<title>Changhing background color</title>

<script language="javascript" type="text/javascript">

function poll() {

//loop through array of click buttons

for (var i=0; i<document.f.rad1.length; i++) {

if (document.f.rad1[i].checked) {

//if a button has been clicked, then assign a background color!

document.fgColor = document.f.rad1[i].value;

//document.bgColor = document.f.rad1[i].value;

}

}

}

</script>

<style>

h2 {

margin: 20px 0 20px 200px;

}

form {

margin: 20px 0 20px 200px;

}

</style>

</head>

<body>

<h2>Changing background colour</h2>

<form name="f">

<input type="radio" name="rad1" value="white" onClick="poll()">white

<input type="radio" name="rad1" value="silver" onClick="poll()">silver

<input type="radio" name="rad1" value="skyblue" onClick="poll()">skyblue

</form>

</body>

</html>

**9. Building a dynamic template - Times table**

<!DOCTYPE HTML>

<html>

<head>

<title>Loops 1</title>

Select ActionSelect AllTry It<script type="text/javascript">

document.write("<h1>Multiplication table</h1>");

document.write("<table border=2 width=50%");

for (var i = 1; i <= 9; i++ ) { //this is the outer loop

document.write("<tr>");

document.write("<td>" + i + "</td>");

for ( var j = 2; j <= 9; j++ ) { // inner loop

document.write("<td>" + i \* j + "</td>");

}

document.write("</tr>");

}

document.write("</table>");

</script>

</head>

<body>

</body>

</html>

**10. Form validation**

<!DOCTYPE HTML>

<html>

<head>

<title>use if else</title>

<script language="javascript" type="text/javascript">

function validate() {

// validate firstname

if(document.f.fname.value == '') {

alert('enter first name');

return;

}

// validate surname

else if(document.f.lname.value == '') {

alert('enter last name');

return;

}

// validate email

else if(document.f.email.value == '') {

alert('enter email');

return;

}

// Set up a DISCLAIMER: confirm submit

subNow = confirm('do you want to submit now');

// submit

if(subNow) {

document.f.submit();

document.f.fname.value = '';

document.f.lname.value = '';

document.f.email.value = '';

}

else {

return;

}

}

</script>

</head>

<body>

<h2>booking form</h2>

<form name="f" action="#">

<p>First Name: <input type="input" name="fname" value="" /></p>

<p>Surname: <input type="input" name="lname" value="" /></p>

<p>Email: <input type="input" name="email" value="" /></p>

<div style="margin-top:20px;"><input type="button" value="submit" onclick="validate()" /></div>

</form>

</body>

</html>

**11. Use break() command to stop loop**

<!DOCTYPE HTML>

<html>

<head>

<title>Break</title>

<script type="text/javascript">

document.write("<p><b>Example of using the break statement:</b></p>");

var i = 0;

for (i=0; i<=10; i++) {

if (i==3){break}

document.write("The number is " + i);

document.write("<br />");

}

</script>

</head>

<body>

</body>

</html>

**12. Use continue() to interrupt loop**

<!DOCTYPE HTML>

<html>

<head>

<title>Continue</title>

<script type="text/javascript">

document.write("<p><b>Example of using the continue statement:</b><p>");

var i = 0;

for (i=0; i<=10; i++) {

if (i==3){continue}

document.write("The number is " + i);

document.write("<br />")

}

</script>

</head>

<body>

</body>

</html>

**13. Using call() method with object properties**

<html>

<title>JavaScript using Call and Apply</title>

<script>

var person1 = {name: 'Marvin', age: 42, size: '2xM'};

var person2 = {name: 'Zaphod', age: 42000000000, size: '1xS'};

var say = function(greeting,typeg){

alert(greeting + ', ' + this.name + ' ' + typeg);

console.log(greeting + ', ' + this.name + ' ' + typeg);

};

say.call(person1, 'Hello','morning');

say.call(person2, 'Goodbye','Aurevoir');

</script>

<body>

</body>

</html>