JAVASCRIPT - CREATING A TOC

Problem specification - Adding a Table of Contents.

The aim is to be able to show a complete novice to HTML, how to add a Table of Contents (TOC) to a page inside a pair of div tag (with id="toc") provided for this purpose. Initially the TOC should list all H3 headers. and each item in the TOC must include a hypertext link to the relevant (h3) paragraph. Then having established a procedure, to use javascript to achieve the same result automatically.

Prepare a set of (plain language) instructions which detail the steps to be followed. The procedure you develop should not just apply to this page for example, but any similar HTML page, that has provision for a TOC in the form of a placeholder (i.e. <div id = "toc" > ... </div>

Pre-requisites

A knowledge of HTML anchors and anchors within a page.

Step 1 - Building a TOC manually

Make a copy of this page and place your plain language instructions for building a TOC here so a complete novice to HTML can follow.

> The objective of this exercise is to find a **client side** (Javascript) solution to this problem; as the only tools required are a text editor and a browser. The procedure you develop to achieve this, will be identical, to a **Server side** or PHP coding solution, only the programming syntax will be different; which is why **pseudo code** so useful, because it is not language dependent.

Now either make a copy of the instructions above, or use HTML strikethrough ()<s>) tags to eliminate all unnecessary words, without loosing the meaning. The object here is to reduce the instructions to a minimum list of words, which now becomes your structured English / pseudo code version.

Process variables

Next try and identify any **variables** that you think this procedure has to make use of. At the start of a project, its only likely you can identify more than a few of the variables required. They will however become apparent as you progress.

When developing any procedure, its easier if you imagine it is already running. Don't worry about the initial **starting** or**ending** conditions, these are generally easier to define once you have a basic procedure in place. For example in this case do you start at the top, or bottom, of the web page? How do you, or more precisely the computer know when the procedure has finished? Obvious questions perhaps, but to a computer it doesn't know the answers unless you tell it!

There may be circumstances where you need to ask a question (selection). Deal with the main path first, then go back and deal with the exceptions.

Finally if you need to get any input from the user, how do you know what they have entered is valid? Again leave these kinds of issues until last.

Place a brief description of any variables here. You may choose to add to this list later. Your pseudo code should now consist of just *generic* words and symbols (e.g. +, = etc.) and variable names. It should not contain any punctuation, with the possible exception of brackets, curly or otherwise. To avoid ambiguity, each statement (or action) should be on a separate line.

Additional tags required

Make a list of any additional types of HTML tags here, (other than P and H3), that are essential to making the procedure work, and highlight any tags which may be unfamilar to a complete novice.

This concludes the first part of the tutorial. You should now be ready to move on to the coding, but before you do, lets deal with some specific things you need to know.

Step 2 - The HTML interface and getting information from the document

There are two main ways of getting information about the elements on a webpage. You're going to be using both, in this tutorial. These are

- document.getElementById('*element*');
- document.getElementsByTagName('*tagName*');

The method getElementsByTagName('*tagName*'), gets a list of all child elements attached to the nodes with the specified tag name. It may appear to be an array, but it is actually an DOM NodeList object. (Don't worry about what that means, its an Object, that's all you need to know at this stage.

Objects

Objects have properties and methods. To give you a simple example, in an object orientated world you are an instance of a person object. One property you have is your name. You also have a method for calculating your age, based on today's date, though unlike a computer it might take you a bit of time to calculate it down to a number of days. Javascript has a number of built-in objects so as well as getting to grips with the syntax of the language, learning how to use these objects effectively is at the heart of program development.

The **dot notation** is used to get an objects property or call an objects method. E.g.

```
objectName.propertyName
objectName.methodName(optional arguments)
```

The first method, we are going to make use of is

```
var tag = document.getElementsByTagName('*');
```

By and large Javascript properties and methods use the camelback notation. Capitals denote second and subsequent words in a function name (hence camelback), so the line above must be typed exactly as shown.

This line declares a variable called *tag* and sets it equal the output from the document object method. The tag name in this case is '*' a **wildcard**, meaning get all of them!

The next thing we need to do is find out how many tags we have got. So we'll create a new variable (lets call it total) and use the objects property length, to find out how many. Now you can use the alert box to see how many.

```
var total = tag.length; alert(total);
```

If I want to pick off a particular tag, say number 5, then instead I can use the tagName property to find out which one it is like this. Make sure you actually have a tag 5 first though.

```
alert(tag[5].tagName)
```

Note: The first tag is not 1, but zero!

Done that? Now we will list them all, (using a **for** loop to get each tag in turn)and put the output in the div box with id = 'toc'. Now the number 5 has become a variable, so we need to give it a name. For simple iterators a letter will do, but normally variables should be given meaningful names.

Notice in the following I've lost the alert box. Modern browsers are not keen on getting to many alerts. I have also wrapped this little procedure (or method) in a function called test().

```
function test()
{
   //program to get all tags on a page
     var tag = document.getElementsByTagName('*');
     var total = tag.length;
     var output = ""; //declare an output variable
     for (var i = 0;i<total;i++)
     {
   //read this as, (new) output = (old) output plus tagName plus a
space.
   output = output + tag[i].tagName + " ";
     }
   //get specific tag by its id and set its innerHTML property
   //to the variable output
     document.getElementById('toc').innerHTML = output;
}
```

The **for** statement can be read like this. Set var i = 0. if i less than total (TRUE), then carry out tasks inside the curly braces. Next increment i, the i++ bit, and repeat until the condition i < total is FALSE. It relies on you knowing how many times to repeat this task, hence the reason for getting the length first.

Don't just copy and paste the solution. Use the Template.htm file and type in the necessary HTML and Javascript code. Then try and run it. If it doesn't work, Welcome to the world of debugging. The problem almost certainly we be a typo. Javscript doesn't give much help when it comes to debugging, it either runs or not, so don't type in a page of code before running it, because you'll be commenting it out again to try and isolate the error.
Comment out chunks of code is a great debugging tool. Some things to note.

- It doesn't matter what you call your Javascript function, provided it not a reserved word.
- Make sure the function you are calling from the HTML button exists.
- Javascript is event driven. When you click the form button, it runs the function you called
- The code block for the function starts and ends with curly braces { }.

- Commands (statements) are separated by semicolons. Much like sentences are separated by full stops.
- Javascript is case sensitive, so spelling and case are important
- Anything between double or single quotes is a **string**. It doesn't matter which type of quote you use. If you need to include quotes, inside quotes, then make the outer quote (say) a double, and the inner quotes single quotes. Make sure they match. Mismatched quotes are a very common source of error.

What you have learnt

The concludes the second part of this tutorial. You should now have some sample code that gets a list of all the HTML tags on the page.

- You have used an object (document) and three of its methods. The third one is innerHTML.
- You have declared 3 variables to capture information.
- You have used **repetition** a key programming structure in any language. The **for** is just 1 of 3 main ways to do this.
- Seen how to concatenate strings using the plus symbol.
- Less obviously you have seen how to mix **strings** and **variables** in the line
  output = output + tag[i].tagName + " "; tag[i].tagName is the variable bit.
- i++ is shorthand for (new) i = (old) i + 1;
- Output = output + *something* is such a common structure, it to has it's own shorthand version
  i.e. output += tag[i].tagName + " ";. Be careful with this to start with, miss the + sign off and the output won't be what you expect!
- Note: the list of tags does **not** include the closing tags

---

Step 3 - Selection of the H3 tags and adding links

In the second part of this we discovered how to access and print the name of each tag on the page. This culminated with the following Javascript function.

```
   function test()
   {
     var tag = document.getElementsByTagName('*');
     var total = tag.length;
     var output = ""; //declare an output variable and set it to
 empty string
     for (var i = 0;i<total;i++)
     {
```

```
     //read this (new) output = (old) output plus tagName plus a
space.
       output = output + tag[i].tagName + " ";
    }
  //get specific tag by its id and set its innerHTML property
  //to the variable output
    document.getElementById('toc').innerHTML = output;
  }
```

Pseudo code description to filter H3 tags

Now we need to learn how to sift out just the h3 tags using **selection**. As we step through each tag in turn, the question we need to ask is

Is this an h3 tag ? yes (true) or no (false) and in each case, what do we need to do?

If the answer is true we need to add the tag to the TOC along with its hypertext link. Also we must add an anchor tag to the h3 heading itself.

If however the current tag isn't an h3 tag, then there is nothing to do, so we can move on to the next tag.

Determining the layout of the TOC

We also need to decide how the TOC is to be laid out. The simplest solution is to use an unordered list. The format for each item in the list will be, (using pseudo code)

```
    define output variable = "<ul>"
    //now for each tag append
    "<i>the hypertext link and h3 heading string</i>"
    //lastly append the closing ul tag
    "</ul>"
```

Converting the pseudo code fragment to Javascript

More formally the Javascript syntax to do this is stated as

```
    if ( tag[i].tagName == 'h3')
    {// condition is true. Copy tag to TOC and add the links
       var hlink = "<a href = '#q" + i + "' ><h3>"; // construct the
opening tag
       // add the text from the h3 heading and the closing tags
```

```
      hlink = hlink + tag[i].innerHTML + "</a></h3>";
      //next add the tag to the TOC list
      output = "<li>" + hlink + </li>";
      //now create the anchor in the same way
      var anchor = "";
      tag[i].innerHTML = anchor + tag[i].innerHTML;
   }
```

Putting it all together

The above code now replaces the code inside the loop of our original procedure. The result is

```
   function test()
   {
      var tag = document.getElementsByTagName('*');
      var total = tag.length;
      var output = "<ul>"; //declare an output variable and set it
to empty string
      for (var i = 0;i<total;i++)
      {
         if ( tag[i].tagName == 'h3')
         {
            var hlink = "<a href = '#q" + i + "' >";
            hlink = hlink + tag[i].innerHTML + "</a>";
            output = output + "<li>" + hlink + "</li>";
            var anchor = "<a id ='q" + i + "' ></a>";
            tag[i].innerHTML = anchor + tag[i].innerHTML;
         }
      }
      //finally write the accumulated output to the div 'toc'
      document.getElementById('toc').innerHTML = output + "</ul>";
  }
```

This is a working algorithm or javascript function. Add the function to the script tags on this page.

What happens if the user presses the form button more than once? *Just see see what happens*, if they do. How might you fix this?

If you try to view page source to directly see the effect of adding HTML code using javascript, you will only see the original source for the page. From the browsers file menu use the ***save as*** option to save a copy of the modified page, to see what is going on.

Is this the most efficient way of achieving this task?. From step 2 you will have seen that most of the tags are not H3 tags. This probably isn't an issue, even on much longer pages. The criteria perhaps is, at what point does it start to impact on the user?

There is in fact a better way to do this, and the clue is in the command

```
    document.getElementsByTagName('*');
```

Building a better TOC

Why did we use a wild card when we know which tag we want. Replacing the wild card'*' with 'H3' will get just the H3 tags. so no need to filter then out, it has already been done for you. Compare the version below with the algorithm we developed. What are the differences?

The original algorithm is still perfectly valid, because it doesn't target any particular language. Only when the target language has been chosen, can potential improvements be explored. Even so, without a radical rethink, the outline procedure, generally remains the same

```
  function toc()
  {
     var tags = document.getElementsByTagName('H3');
     var toc = "<ol>";
     for(var i = 0;i<tags.length;i++)
     {
   toc += "<li><a href ='#q" + i + "' >"
          toc += tags[i].innerHTML + "</a></li>\n";
   var anchor = "<a id ='q" + i + "' ></a>";
   tags[i].innerHTML = anchor + tags[i].innerHTML;
     }
     document.getElementById('toc').innerHTML = toc + "</ol>";
     }
```

String methods anchor(name) and link(url)

The anchors and hypertext links created above were constructed manually; too show what is going on. There are however string methods to do this as shown in the following version.

```
    function toc()
    {
   var tags = document.getElementsByTagName('H3');
   var toc = "<ol>";
```

```
    for(var i = 0;i<tags.length;i++)
    {
            toc += "<li>" + tags[i].innerHTML.link("#q" + i) +
"</li>\n";
        tags[i].innerHTML = tags[i].innerHTML.anchor("q"+i);
    }
    document.getElementById('toc').innerHTML = toc + "</ol>";
     }
```

This version has been included in the script tags for this page. Change the form button to call toc() instead of test, (or better still add a second button) and compare the results. Is there any appreciable difference in the time it takes?

From a maintenance point of view, clearly the more compact the code, the easier it is to understand. The functions have been stripped of any comments, as you would do in any production version, but a copy, fully annotated, should be kept for future reference. Because simply from looking at the code, its not immediately apparent, what is going on. By merging the plain language, (pseudo code or structured English) version of the algorithm with the Javascript code, as comments. both complement one another

Creating an external link to the Javascript

Once you are satisfied everything is working, create a separate page to contain the scripts you develop. In this way you can build up you own library of useful scripts. They can then be easily linked to any page by placing a link in the head portion of the page.

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

Summary

In this tutorial you have been introduced to both **repetition** and **selection**, the two key structures of any procedural programming language. You have seen how, even with a basic knowledge of Javascript, how by using **strings** and string methods, a web page can be modified to include a Table of Contents (TOC). The tutorial has shown that, as well as learning the Javascript syntax, a better understanding of the functions available, can help to more efficient and compact code. Making use of these functions also illustrates an important difference between structured English (i.e. what has to be done) and the final code (i.e. how it is actually achieved).

The original algorithm described how to get all the tags on a page, then filter out just the H3 takes we wanted. This whole task was accomplished using the javascript function getElementsByTagName('h3').Similarly the addition of anchors and hypertext links was also completed using two corresponding string methods.

In building a TOC, the specification placed no constraint's on the original document, and is itself, possibly a simplification of a real world requirement, that perhaps need to take account of all headers, through H! to H4 say. There maybe other scenarios when specifying some initial preconditions, makes the programming task much simpler. Any such preconditions should be included in the specification, and agreed with the owner of the specification (the user).

There may be a case in this instance for using DIV tags, to separate each section of the document defined by the H3 tags.

Clearly by using Javascript, the task of adding a TOC to a new page, or adding it retrospectively, will save considerable time. It has also de-skilled this task making it possible for an HTML novice to achieve the same result unaided.