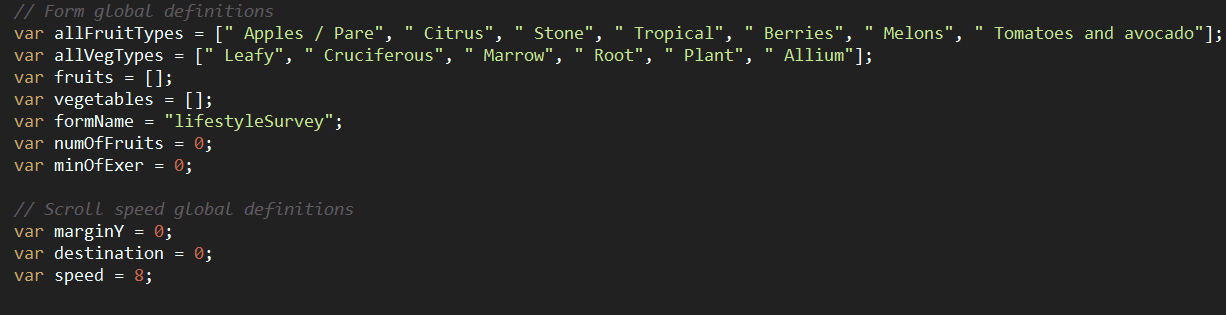
# Good practice

As important as the functionality of the code, is how well it meets common standards and practices, found amongst other scripts. This can include improving the readability for other users, by following common conventions that are accessible by other developers, as well as standardised methods of implementation, that are globally recognised. This report will cover the basis of these practices, as well as how they have been implemented within the different webpages of my website. Code illustrations will be given (where appropriate) to show what the code is doing, and how it follows common good practices.

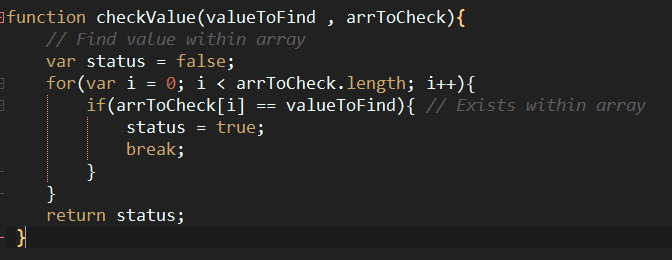
# Script commenting:

The use of comments within code is vitally important in explaining to other developers what is happening within your code, or more importantly how / why it is happening. Any piece of code that is trying to come across as accessible to other developers will have to use commenting; you can think of it as a dictionary for the different aspects of your code. If a developer wants to find out what an aspect does, they look at the comments for their answer.

I have used a variety of comments throughout my scripts; both JavaScript and CSS (can also apply to HTML) as to make sure of following this good practice. It also helps make it easier for other developers (including myself) to keep track of what is going on.

1: JavaScript:

I begin with commenting the variable declarations (shown above), to idealize the grouping that a type might belong within. For example, above shows how I have grouped the two-global variable sets into those used specifically within the form, or those for external purposes, such as scrolling the page. You can continue similar approaches here, splitting the variables into smaller subsets and commenting what they relate to however, it can be clear from good variable names of their use, so this is not always necessary. As a good practice, the use of comments here is important, as it allows other developers (including myself) to easily identify what the variables are going to be used for, as well as where they might be used. For example, looking at the code above, I will instantly know that the variable ‘numOfFruits’ will be used within the form on the Lifestyle Survey page; the name of which implies it will be used to keep track of the number of fruits the user has inputted. I have just found out what a specific variable will be used for within the script, without even needing to look at any of the code (beyond declarations), due to the use of comments.



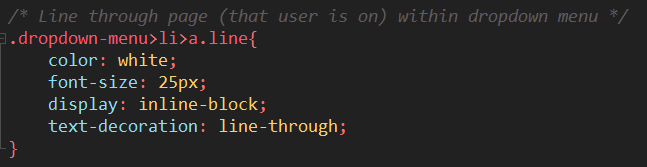
I have continued the use of commenting within the code, to apply within the distinct functions making up the processing of the webpages. For example, above shows a simple function that carries out the task of checking if a value exists within an array, then returning the status (Boolean). We can easily find out the process of the function with the commenting applied. The first comment gives an overview of what the function is designed to carry out, but is distinct enough from the functions name to allow for relevant information; it tells us what it does. Now that we know the functionality of the function, the second comment enforces this by telling the process to achieve the result; it identifies and explains the piece of code which allows the function to carry out the specified task (within the first comment). No other comments are needed within the function, as the user can easily distinguish the other tasks just from reading the code:

* Status begins false
* Loop through every element in list (this is clear just from the code)
* If element exists (from comment)
* Change status
* Return status

Essentially, the comments help fill a gap within the code, where the user might not otherwise be able to tell what is going on from reading the code. It shows good practice adopted within the code, as it drastically improves the readability and expandability of the code. Using them sparingly (where appropriate) is also good practice, as comments are not needed where the user can easily tell what is going on, just from reading the code; there is no point just re-explaining the code in a way that tells nothing new. There are only two comments in the function above, as the rest of the processes can easily be obtained from reading the code, so comments would not be necessary, identifying good practice from them not being used in this situation.

2: CSS:

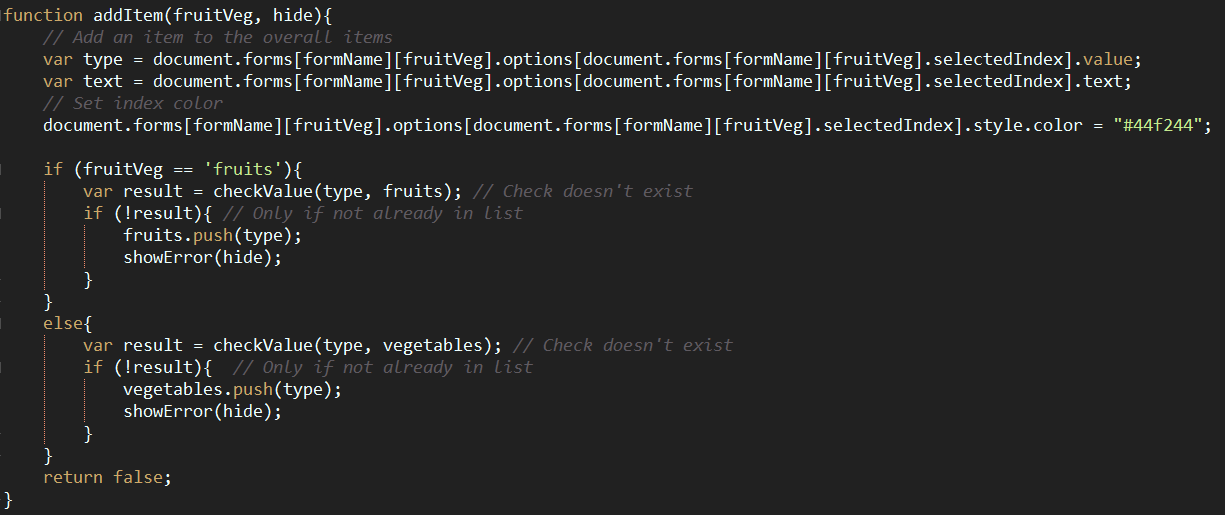
The use of commenting has also been adopted within CSS scripts, to identify good practices adopted within my webpages.



Within the CSS, I used commenting to explain to the user what a specific block of code (styling segment) is designed to style within the webpage. For example, the above segment of code, without the use of commenting, would be underlyingly difficult for any developer to know what it does. Using comments here allows for the code to be related back to specific HTML webpage elements and essentially identified within the CSS. From this comment along we have learnt: what the code does (line-through) and where it will be applied (page element that user is on within dropdown menu). This shows example of good practices within the code, as it helps improve the readability and expandability for other developers, who might be using the code.

# Script indentation:

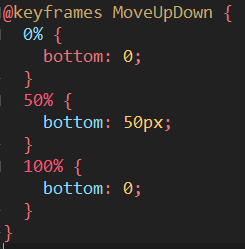
To a degree of importance, script indentation also helps apply readability and expandability within code, as a good practice. Indentation within scripts are an industry standard so would be expected as mandatory with all coding languages; including JavaScript, CSS and HTML. The use of which is self-explanatory; it is when child elements are indented from their parent elements, to help distinguish what code belongs to which parent element, as well as top split the code into identifiable blocks.

1: JavaScript:

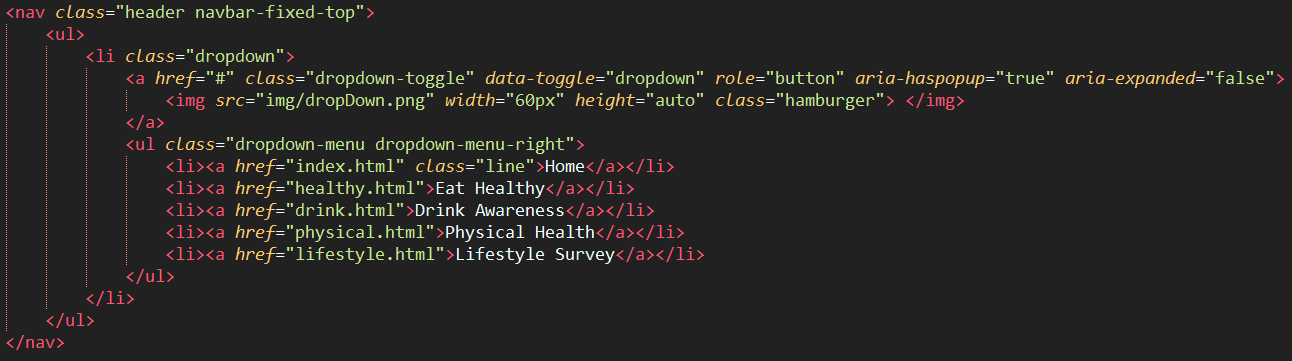
All the JavaScript within my website has been written with code indentation in mind; every function, method and block of code has applied the rules of indentation as a good practice. You can see these rules within the function above, where every child element is indented from its parent element. For example, look at the IF selection statement within the function; you can see how every time a process begins (within a statement), it is indented from the rest of the code. The use of curly braces here help insinuate the code indention, as every time a brace opens, the code is indented, and every time a brace closes, the code is un-indented (returned).

As a good practice, following these rules makes sure that all code is indented appropriately, to allow for improved readability for other developers, and therefore expandability; any code that is easier to read is easier to build upon. You can see this within the function above, as every child element is indented, it is much easier to identify segments of code; I know anything indented within the function, belongs (and is attributed) to the function, and anything indented within statements (such as If statements) belongs to those specific statements.

2: CSS:



Within the CSS, all the rules described as suitable within JavaScript, can also be applied here. The example above shows how their rules have been implemented within the CSS scripts. A parent element (such as a statement); in this case a ‘@keyframes’ declaration, has all its child elements (within the curly braces) indented, to identify that they belong to the specific statement. This is carried over to all the other statements, such as the animation frames within the keyframe; where the process is indented from the frame (as shown above). Therefore, the good practice of indentation is applied within the CSS to allow the same tropes, of better code readability (furthermore, expandability) within the CSS; it just makes it easier for other developers to know what the code is doing, which would be made unnecessarily difficult without the use of indentation (why it is such a good practice).

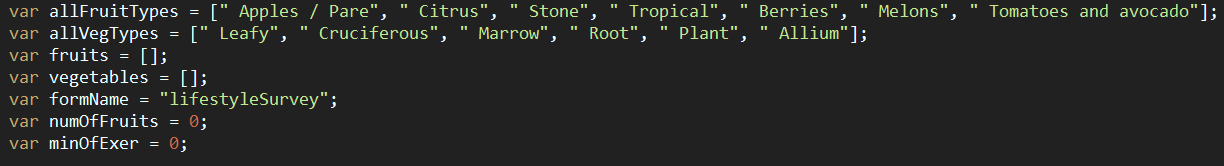
3: HTML:

Evidence of appropriate indentation is also relevant within the HTML scripts. Arguably, the rules of indentation apply the most towards these HTML documents, as you are directly working with child and parent elements, which need to be proportionately indented from one another. For example, the script above shows some elements where these have been directly applied. The <ul> (unorganised list) shows a relevant example as a parent element; each of the <li> (list item) child elements are indented from the <ul> parent, as to show that they are sub-elements of the parent.

You can essentially apply the rules of indentation to any child element within HTML however, sometimes it is not appropriate (mainly not needed) to indent child elements, as it can provide no negotiable benefit. For example, looking at the <a> (hyperlink) child element, within the <li> parent, there is no need to indent here, as its clear the tag belongs within the list item; you can identify where the parent is closed. Good practice is still applied here as the code is just as readable without the need of indentation, although where child elements are not clear, the use of indentation should be applied to allow the benefits of the practice (readability and therefore expandability).

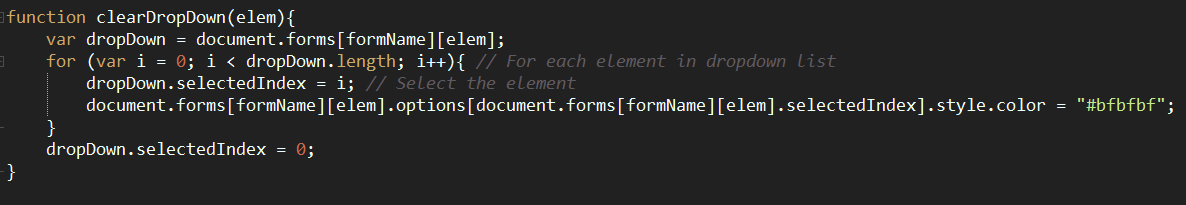
# Script naming conventions:

The referral of script naming conventions relies on the application of appropriate names used for different variables, functions, methods etc. within the different scripts. Therefore, this good practice can be applied to any script that make heavy use of these named types; mainly JavaScript, but to a lesser extent joint HTML and CSS (for Classes and ID’s). We use the naming conventions for these types, as to make sure they are relatable and explicit with the data they will store / the function they will carry out; it lets other developers understand the usage. They are also important to make sure of continuity throughout scripts, with names following explicit rules (set by the developer) of how they are defined, which can be followed throughout the scripts.

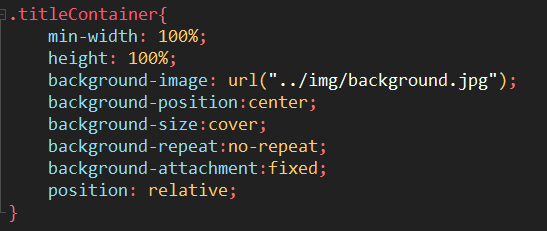
1: JavaScript:

Relating back to the same code examined within the comments, we can now infer the good practices applied to the naming conventions of the different variables. For a start, the use of a CamelCase naming convention has been applied to all variables. This ensues that each ‘segment’ of a name, that might otherwise be contained as a separate word, is defined with a capital, whilst the start of the final formed name (first word) is always lowercase. We can see this applied to the ‘allFruitTypes’ variable; essentially, we have 3 separate words: “all fruit types”, but these are formed into a single string, with the start of every word being a capitol (besides the first word).

Continuity when using a specific naming convention is paramount, as to not confuse other developers (or even yourself) with declarations. You can see evidence from the example above that I have applied CamelCase to all variables, as they all follow the same rules. Therefore, it allows for good practice as it makes it clear what is a variable declaration and tells other developers how they are defined.

Furthermore, the naming conventions applied to the variables also allow them to become explicit in describing and relating to the data they are storing. For example, going back to the ‘allFruitTypes’ variable, we can find out exactly what the variable is storing just from its name. We know it is an array, as the use of ‘all’ denounces the variable as plural; being more than one. We also know the variable stores all different fruit types, as it is clearly described from the ‘FruitTypes’ convention within the name; applying this as ‘all’, we now know all fruit types are stored within the variable. Therefore, just from the name of the variable alone, we have learnt the type is an array storing all fruit types. As a good practice, it is vital to include these naming conventions to describe to other developers how and what the variables are used for; improving the overall readability of the code.

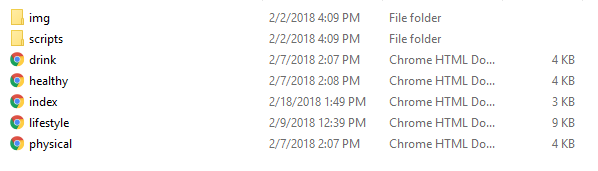
I have also applied similar naming conventions to each of the function names; they all use the same CamelCase formatting, as is ensured continuity (and therefore better readability) within the code. Choosing a suitable name for each of these functions is also just as important as it was with variables. For example, the name ‘clearDropDown’ has been chosen relatively to how well it can describe the functions use. We know from reading the name that the function is designed to clear a dropdown within the form; with the dropdown being the ‘elem’ passed through the parameters. Therefore, just from reading the functions name, we now know what and how the function is used; being a method of clearing dropdowns.

HTML & CSS:

As explained earlier, the prevalence of good naming conventions can also be found within HTML and CSS, often join using Classes and ID’s. These naming conventions are specifically important as they need to provide a link between the two languages (as well as within JavaScript), especially if external scripts are used. For example, the parent <div> above has a class applied to it for styling; ‘titleContainer’. The name of this class follows the same CamelCase convention, to carry the standard across all the documents of the webpage, as a good practice; it makes every named type instantly recognisable. More importantly however, is to make sure the name is relevant to the element it is selecting (to be styled).

The example above shows how the good practice of naming conventions has been implemented, as the name instantly describes the division within the page; “title container” refers to a container that holds the title of the webpage, and this is exactly what the division does. When we look at the styling within CSS, because of the naming convention, it means we know what the styling is applying to; you can look within the CSS, see ‘.titleContainer’ and know that it is styling the container containing the pages title. As a good practice, it helps other developers (including myself) to recognise what and why the code is doing something, such as why and element is styled a specific way. We could also improve the convention further, by stating what the title is; “indexTitleContainer” might show a further improved naming convention, as it further describes the element and its styling.

# Document formatting and organisation:

It may not seem important when implementing, but the use of organisation within a websites files allows for better management and implementation of features, such as images and external scripts. When we refer to formatting and organisation, we are talking about managing the websites files, into specific folders and sub-folders to contain the relevant data. For example, you might have a folder for all images, and then another sub-folder within, just for background images. The same general rules can be applied to all files that make up the webpages of a website.

We can see these rules applied towards by website, as I have grouped the files into several different folders and sub-folders. Such as, I have individual folders for the various scripts and images that the website makes use of. Therefore, only the webpages themselves are found located within the root folder. In terms of management, it makes it easier for me (including other developers) to locate specific files; if you wanted to locate the CSS and JavaScript scripts, then you would navigate towards the ‘scripts’ folder, likewise if you wanted to find the images used, you would navigate within the images (‘img’) folder. It is this improved management, that makes it such as good practice.

Furthermore, I could go even further with this practice, creating more folder and sub-folders to better organise the website (although it may not be necessary for a small site). For example, I could create a folder to organise the different webpages into, instead of just chucking them all within the root folder; especially if you have lots of webpages, where organisation becomes difficult. I could also split the scripts folder into relative subsets, such as CSS and JavaScript folders respectively; if you had more than one stylesheet for each, then this would be an especially helpful practice to help with grouping.

# Implementation of CSS & JavaScript:

When we refer to the implementation of CSS and JavaScript as a good practice, we are generally regarding the use of external scripts, instead of inline / embedded scripts, as these are generally regarded as bad practice. The inclusion of how these external scripts are referred within the webpages, can also be regarded as good practice, as if done properly, can allow for more seamless loading.



Within each webpage, I have used external scripts for the inclusion of CSS and JavaScript, as the primary method of scripting; I have not used any embedded scripts within a <script> tag, as these are generally regarded as bad practice, due to their poor expandability and manageability. When you have lots of webpages using the same styles / JavaScript, it becomes inefficient to re-write the same code for each webpage, when they can instead just be shared across a single external script.

Importing the CSS has been implemented within the <head> tag of each webpage, this way it allows the CSS to be loaded at the beginning of the webpage, so that the styles can be applied to each of the elements within the HTML. Importing CSS at the end of the <body> tag could result in styles not being applied to the HTML elements, as they have already been rendered (it doesn’t iterate), leading to a style less, flat webpage; so is generally regarded as bad practice. Furthermore, I have used the inclusion of the ‘?version’ attribute within the <link> tags ‘href’; the use of which, as a good practice allows the cache of the webpage to be over-written, every time a new version of the CSS document is returned. In doing so makes sure the most recent styling is applied to the webpage, with any old styles (with cache) being overridden, so is generally regarded as good practice.

Unlike the CSS import however, the inclusion of the JavaScript import is carried out at the bottom of the <body> tag, after the webpage has been loaded. In doing so, ensues a good practice, because it means browsers that don’t support JavaScript, won’t have the loading of the webpage adversely effected, as the JavaScript is loaded after the webpage. Furthermore, it allows for visually faster load times within the webpage, as it means the visible webpage loads first, followed by the JavaScript. Even if the JavaScript takes a long time to load, it doesn’t matter (to the user), as to them the webpage is perceived to be loaded, as they can visually see it. This is especially important when dealing with large JavaScript files (or multiple) that do take a long time to load, so is therefore regarded as good practice.

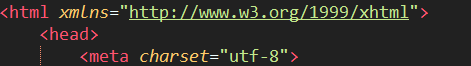
One practice I adopted within my webpages that is relatively bad practice, is the use of embedded CSS styles within curtain elements, such as <img> tags. Doing so means the styles are spread around the place, making organisation and manageability of the CSS inherently more difficult; it essentially just shows lazy implementation. Instead, I should put these styles in the external script (like the rest), and use a class to relate the <img> to the style (the current classes applied don’t do this).

# Implementation of XHTML standards:

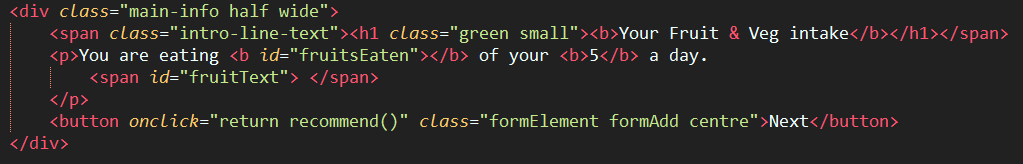
Finally, the implementation of different XHTML standards can arguable be described as the most important, when it comes to implementing good practices within webpages. These standards typically refer to; the inclusion of a <!DOCTYPE> tag, XML namespace definition, all elements and attribute names must be lowercase, all empty elements closed, all attributed values quoted and the representation of specific HTML tags. The inclusion of all transforms a webpage from HTML to XHTML as a good practice.

Doctype:

The inclusion of the <!DOCTYPE> attribute is defined as a good practice within the XHTML standards, as it makes sure the webpage documents are defined. The <!DOCTYPE> tag is always included as the first element within a webpage, as to make sure the document is defined to the browser, before the scripts are read. This way, it gives instructions to the browser as to what Markup language is used, as well as the version, so that is can prepare the rendering of the webpage. Each of the webpages within my website include this tag, defining the documents as ‘html’ to tell the browser what language is used, therefore following the good practice and allowing for proper implementation of XHTML standards.

XML namespace:

The use of the XML namespace within the <html> tag is important for defining the location of the document, which contains the naming for the XML application. We place the XML namespace within the ‘xmlns’ attribute, passing a link to the path of the XML document for inclusion. Essentially, if you want to document to become XHTML, you need this attribute. Unfortunately, the use of the XML namespace within my website was not properly represented, with the ‘xmlns’ attribute missing from <html> tags; it shows bad practice within the webpages for it to not be included. Therefore, I need to make sure to include the XML namespace next time (such as the example above), to make sure the XHTML standards are fully met, as a good practice.

It is to be noted however, that although the XML namespace was not defined within my <html> tags, it doesn’t mean the webpage can’t comply as XHTML, as a default namespace is defined by the browser and added to the <html> tag if not included; allowing the webpages to still comply as XHTML. Although, it is still generally bad practice to not include the definition.

Elements and attributes must be lowercase:

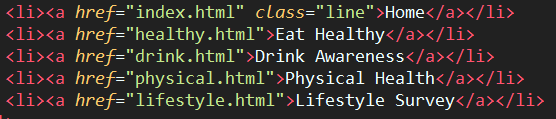
The next XHTML standard defines that all the HTML elements must be lowercase, and the attributes within these elements must also be lowercase. Generally, this is the easiest good practice to ensue within a document, because you will automatically write all elements and attributes as lowercase, as it is just how that have and always will be defined. The example above shows the practice within my webpages; you can see how each of the element tags are all defined as lower case, such the same as the attributes within. For example, looking at the <button> element, we can see the specific attributes of ‘onclick’ and ‘class’ are both lowercase; following the XHTML standards and ensuring good practice.

Empty elements closed:

The representation of closing empty elements is an XHTML standard that is often forgotten, especially with the inclusion of <br> tags. However, it is still required and defined as good practice when creating webpages, as it means all elements within a document are closed, regardless of if they are empty or not.

Within my webpages, the only empty HTML element commonly used was the <img> tag. Above shows two methods of implementation that comply with the XHTML standards, however one is decidedly a better practice than the other. The first implementation method is the type I used within my webpages; closing the tag as </img>. Although doing so does comply with the XHTML standards, it is not very well supported using HTML 5, so could lead to errors or issues during the rendering of the webpage (though it’s still better than an unclosed element). The ideal way of closing an empty tag would be to use the second method; adding a backslash to the end of the tag. It follows both the XHTML and HTML 5 standards, so is generally regarded as a better practice; it should be included within my website, if changes are made.

Furthermore, although unrelated to the standard, the use of the <img> tag within my webpages did not include the ‘alt’ attribute, to specify an alternative description for the image; I simply forgot to include it. Therefore, I also need to make sure this attribute is included within all the <img> tags, as to follow the good and standard practice is followed throughout the website.

Attribute values quoted:

Another easy XHTML standard that was implemented within my website, is the use of making sure all attribute values are quoted. This simply refers to putting the values of any attribute within quotes, whether they are styles, links, classes or JavaScript; they must always be within quotes. I made sure the standard was thoroughly followed throughout each webpage, with the example above showing how it has been implemented; the classes and hyperlinks for <li> elements for example, are all quoted; it makes it easier for other developers (including myself) to see what it a value, and what is an attribute, especially important without the IDE highlighting.

Representation of specific HTML tags:





The final good practice represented as a XHTML standard, is the use of specific HTML tags within each webpage; these include <html>, <head>, <title> and <body> tags, which are compulsory across all webpages. I made sure these tags where properly implemented across all webpages, as to follow the XHTML standard; above shows the typical implementation methods for each of these elements. The <html> tag is important for telling the browser that the document is of HTML type; the <head> is important as a container for all relevant ‘head’ tags which tell the browser what to do, being those which loads before the webpage is rendered (such as scripts, styles, titles etc.). Therefore, we place the <title> tag as a child of the <head>, to tell the browser the name of the document, to be displayed within the browser itself (on the tabs). Finally, the <body> tag contains the bulk of the webpage, including all the visible elements, and those which are rendered to the browser. It almost so common to automatically include these tags, that they are definitely required as good practice, as well as a required XHTML standard.