Welcome to Apps from Scratch – Getting Started (version 2)

1. **Overview**

This document is a brief introduction to the *Apps from Scratch* group and to the scope of software applications that we plan to develop. Also included is a brief introduction to mobile apps, webpages and Google Forms & Sheets.

Over time and with feedback from members, this document will be updated.

For ease of reference, the content is structured as follows ::

**Introductions**

1. Introduction to *Apps from Scratch*
2. Introduction to Mobile Apps, Webpages and Google Forms & Sheets

**Appendices** :

1. Appendix 1 shows the basic software framework for most mobile apps and webpages
2. Appendix 2 describes how to set up the code for mobile apps and webpages on your device and how to run the programs.
3. Appendix 3 gives is a set of example programs for mobile apps and webpages. These programs demonstrate some simple applications to get you started. The programs build incrementally to show how additional code can extend the functionality of the program.
4. Appendix 4 gives a set of example applications for Google Docs and Google Sheets.
5. Appendix 5 (IN PREPARATION) describes how to how to set up the code for Google Docs and Google Sheets on your device and how to run the programs
6. Appendix 6 (IN PREPARATION) describes how to use *Responsive Design*
7. Appendix 7 (IN PREPARATION) shows our current list of website tutorials that will help you further.
8. Appendix 8 (IN PREPARATION) gives a brief glossary of terms that are used within this *Getting Started* document and elsewhere. Terms that are used in this document that are defined in the glossary are italicised.

**Introductions**

1. **Introduction to *Apps from Scratch***

*Apps from Scratch* is a self-help group of members formed to share learning and experiences gained by developing software applications, including mobile apps, webpages and Google Docs to run on a wide range of digital devices.

1. **Introduction to Mobile Apps, Webpages and Google Forms & Sheets**

**Mobile Apps and webpages**

Mobile apps and webpages run on many different devices (or *platforms*), such as mobile phones, tablets, laptops, PCs, Desktops and TVs. In the main, these devices for most users are either Apple (iPhone and iPads), Android (phones and tablets, Windows (phones, laptops, PCs and Desktops) and any TV with HDMI or computer input.

All applications that run on many different devices uses the same software programming languages. To a large extent, the type of device makes very little difference to the design and coding required. Having said that, there is a need to include code that ensures that the intended display of data and graphics automatically takes into account the dimensions of the screen of a range of devices. This should avoid displays being cramped on small devices like phones and alternatively being tucked away in a corner on larger devices like PCs and TVs. This is known as *Responsive Design*. See Appendix 6

**Google Forms & Sheets**

Goggle offers a suite of on-line applications to match Microsoft Office applications including Excel and PowerPoint. At this stage, *Apps from Scratch* uses Google Forms and Google Sheets. These are cloud-based applications and are stored on Google Drive. For more details how to develop and run Google Forms and Sheets, see Appendices 3 and 4. Google Sheets uses very similar formulas to those in Excel, but Google Sheets goes further by extending functionality by using the same software tools as used in mobile apps and webpages.

**Appendices**

**Appendix 1 Basic software structure for most mobile apps and webpages**

There are three programming coding tools used to build and display text and graphics on mobile phone apps and webpages : :

* **HTML** – Hyper Text Markup Language
* **CSS** – Cascade Style Sheets
* **JavaScript**

All applications require :

* **Framework**
* **Format**
* **Function**

**HTML** provides the **Framework**. **HTML** encloses sets of **HTML** code, CSS code and JavaScript code. When complete with **HTML** code, **CSS** code and JavaScript, the **HTML** **Framework** informs the device that an application is ready to run to run.

**CSS** provides the **Format** such as the screen layout, colours, spacing, font types etc.

**JavaScript** provides the **Function**. Of the three coding tools, **JavaScript** is the only actual programming language. **JavaScript** code does the work like calculations and comparisons test (e.g. If *Expenses* is greater than *Income*, display *Overdrawn*) etc.

Note that all mobile apps and webpages will run with **HTML** and **JavaScript** only, but by including **CSS**, the look and feel of the application can be enhanced considerably

**<!DOCTYPE HTML>**

**<html>**

**<head>**

**<title> ----- Application title goes here ------- </title>**

**<style>**

**------- CSS code goes here ----------**

**</style>**

**</head>**

**<body>**

**<script>**

**-------- JavaScript goes here ----------**

**</script>**

**</body></html>**

**Appendix 2 How to set up the code for mobile apps and webpages on your device and how to run the programs.**

The two approaches used by *Apps from Scratch* pivot on the *code editor* that is used., e.g. Notepad and VisualStudio. Notepad is used for simple programs using a basic HTML / JavaScript model. New developers may find Windows *Notepad* suitable (initially). See Appendix 3.1..

For more complex programs using the full HTML / CSS / JavaScript model, experienced developers may prefer to deploy *VisualStudio* SeeAppendix 3.2.

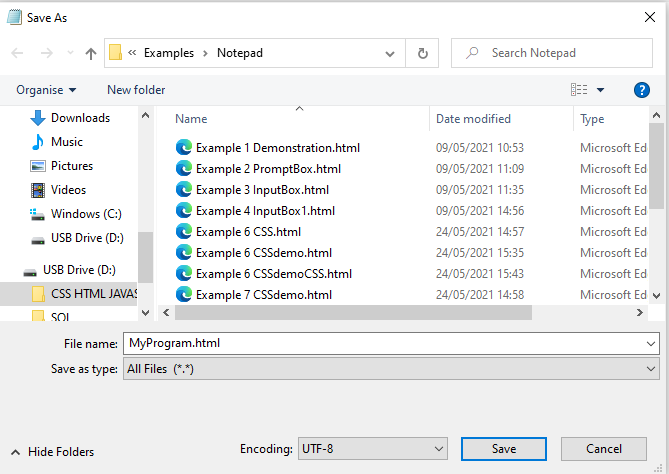
Appendix 3.1 Notepad

Programs saved in Notepad can incorporate the full HTML / CSS / JavaScript model, but it is perhaps more appropriate for simple programs. *Notepad* is included in the standard suite of Microsoft Office suite. It makes for a very straightforward start for new developers.

While Notepad offers basic word processing, it’s main disadvantage is that, unlike *VisualStudio*, it does not include a predicted text facility. More details are given in Appendix 2.2 Visual Studio

To compose and save your code using *Notepad*

* Open *Notepad*
* Compose your code
* Save your file with the file extension of .html and as type All files (“.”). As per the example MyProgram.html in Panel 1



Once saved, you can run your program from Notepad by clicking on the file in the directory

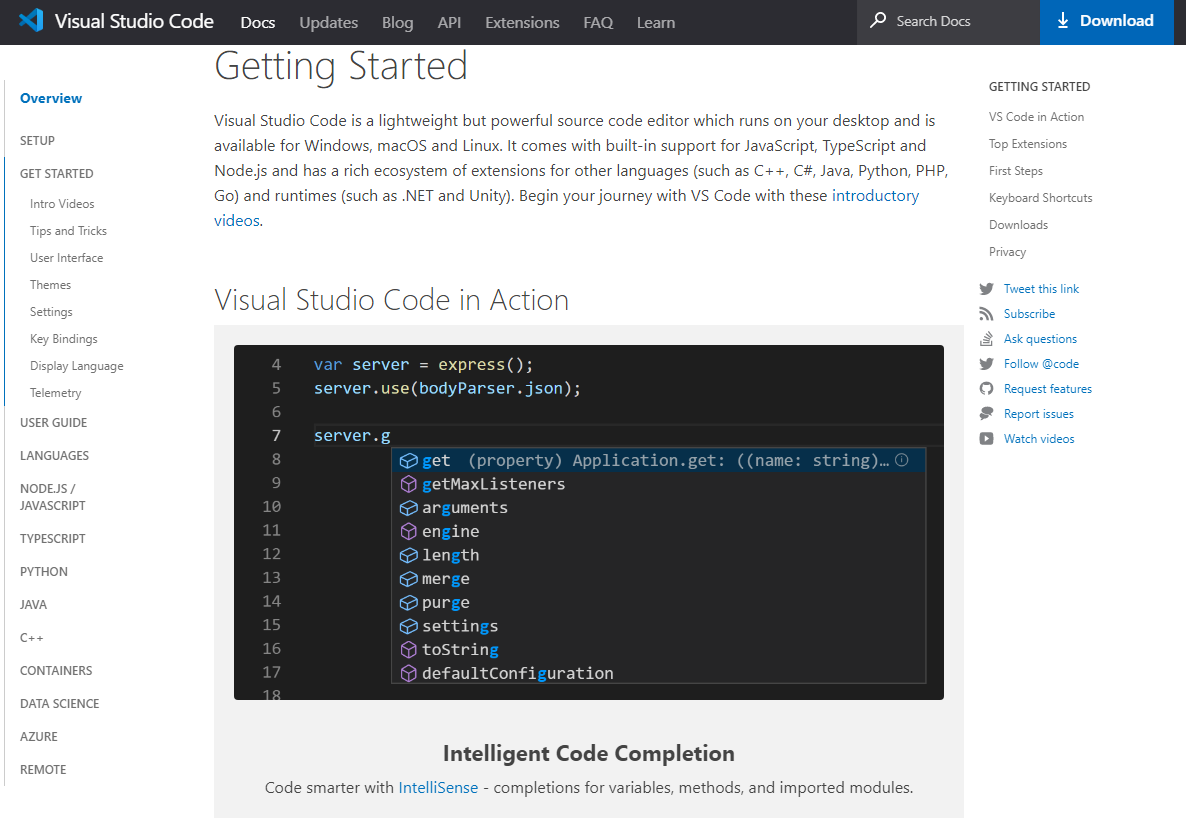
Your browser will open and will run your program (Fingers Crossed!)

**Appendix 2.2 Visual Studio**

Visual Studio Code is an editor designed to create and amend source code. It is more powerful than Notepad, for example providing colour coding for different language elements, auto-completion prompts and drop-down lists of likely entries, and some indications of errors. We recommend using Visual Studio Code as your source text editor as, in addition to its many useful features, it will enable us to share tips, techniques and experiences of using it between members of the group.

It works well with HTML, CSS and Javascript, and also works with a number of other programming languages.

It can be downloaded for free for Microsoft Windows, Apple iOS and Linux. If you go to web page **code.visualstudio.com/docs/**, you will see a brief introduction, links to tutorials including some introductory videos, and a button to download the application for your device – see the screen shot below.



When installing Visual Studio Code in Windows, you can set an option to “Open with Code”, which enables you to open an HTML document directly in Visual Studio Code, by right-clicking the document entry in Windows Explorer: one of the options is then ‘Open with Code’.

We are at very early stages in working with Visual Studio Code, so not doubt we will find many other useful features as we go along. Please contribute with your experiences!

**Appendix 3 Example programs for mobile apps and webpages**.

These example programs start with Program 1 that uses a basic formula to convert 20° Centigrade to the equivalent in Fahrenheit.

Programs 2 onwards add additional functionality (inputs, outputs and processing) incrementally. This should demonstrate how additional code extends the look, feel and function of programs.

Note that HTML code is shown in red and the JavaScript code is shown in blue. (CSS code will be introduced later)

**Program 1 – Converts 20°**

All your programs should include this line of code to maintain *Responsive Design*

There are two variables : cTemp and fTemp

cTemp is used to store the value of 20

fTemp is used to store the result of applying the formula ((cTemp \* 9) / 5) i.e. ((20 \* 9) /5) = 36

<!DOCTYPE HTML>

<html>

<head>

<meta name=”viewport”content=”width=device,initial-scale=1”>

<title> Temperature 1</title>

</head>

<body>

<script>

var cTemp = 20

<br> is used to place the second document.write on to the next line

var fTemp = ((cTemp \* 9) / 5) + 32;

document.write("Temperature in Celsius: " + cTemp + " degrees <br>");

document.write("Temperature in Fahrenheit: " + fTemp + " degrees");

</script>

Program Result

To show on screen the value that is stored in the variable cTemp and also the result that is stored in the variable fTemp, the program uses two document.write statements comprising three components:

1. Document.write (………..)
2. A phrase in double quotes before and after the value
3. + sign *to concatenate* the three components

</body>

**ResultREsult**

</html>

**Program 2 Adds a Prompt Box to Program 1**

<!DOCTYPE HTML>

<html>

This program is based on Program 1. It includes an on screen button, that is labelled, “Display Prompt Box”

When the button is clicked (i.e. selected) , the function prompfFunction is executed

<head>

<meta name=”viewport”content=”width=device,initial-scale=1”>

<title>Temperature 2</title>

</head>

<body>

<p> Click the button to display a Prompt Box.</p>

<input type = "button" id=btn1 onClick="promptFunction()" value="Display Prompt Box">

<script>

function promptFunction()

{

Note that all functions are enclosed within an open and closed braces

The script is the same as in Program 1, except that the script it is executed only when the button is clicked

var cTemp = prompt ("Enter Temperature"); // temperature in Celsius

var fTemp = ((cTemp \* 9) / 5) + 32;

document.write("Temperature in Celsius: " + cTemp + " degrees <br>");

document.write("Temperature in Fahrenheit: " + fTemp + " degrees");

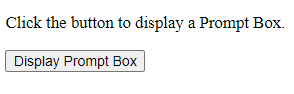
}

</script>

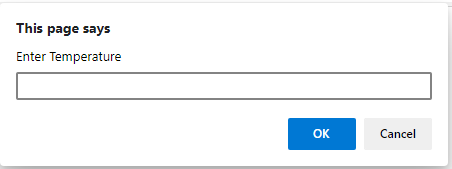
</body>

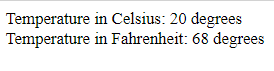
</html>

**Program 2: Results**



1. Program 2 opens with instructions to click on the button to bring up a Prompt Box:
2. Click on the button to show the Prompt Box





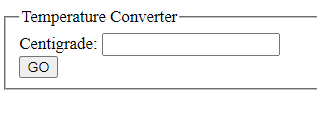
1. Enter 20 in the Prompt Box and click OK. The result should be the same as produced by Program 1

Run the program with different values of Centigrade and check the results the conversion formula manually.

**Program 3**

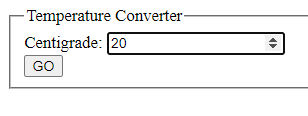
This program takes the temperature conversion further. It opens with an input box that is used to accept input and a button to be pressed to get results. The data flow within the code is indicated by arrows. The program functionality is explained followed by how the code works.

Program 3 opens with an input box, a label and a button. All three are set in a frame.

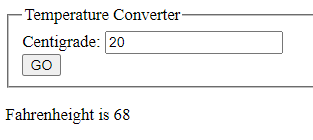


* The frame is labelled : Temperature Converter
* The label for the input box is Centigrade
* The text on the button is GO
* The framing of on-screen data is called a fieldset (See program)
* The label of the frame (i.e. fieldset) is called a legend (See program)

Enter 20 in the input box



Click GO to get a result



The result is given within a *string*

For 20 Centigrade, the program should give the same result as programs 1 and 2.

Run the program with different values of Centigrade and check results using the formula manually

**Program 3 (Continued) Data Flow**

<!DOCTYPE HTML>

<html>

<head>

<meta name=”viewport”content=”width=device,initial-scale=1”>

<title>Temperature 3</title>

<style></style>

</head>

<body>

<fieldset>

<legend>Temperature Converter</legend>

<label for "Centigrade">Centigrade:</label>

<input type="number" select id = "**CentigradeIN**" /><br/>

<input type="button" value="GO" onClick="**myFunction**()" />

</fieldset>

<script>

function **myFunction**()

{

var **cTemp**=(document.getElementById("**CentigradeIN**").value);

var **fTemp** = ((**cTemp** \* 9) / 5) + 32;

document.getElementById("**resul**t").innerHTML="Fahrenheight is "+ **fTemp**; }

</script>

<p id="**result**"></p>

</body>

</html>

**Program 3 (Continued) How the code works:**

The program is informed that :

* the *data type* to be input into the input box is a **number.**

See (<input type="**number**" select id = "CentigradeIN" /><br/>)

* the input box is identified with a **selector id**
* See (<input type="number" **select id** = "CentigradeIN" /><br/>)

When the button is clicked, the function called **myFunction** is executed.

See (<input type="button" value="GO" onClick="**myFunction(**)" />

**myFunction**

Line 1 of myFunction, after the opening brace, assigns a value to variable **cTemp.**

See (var **cTemp**=(document.getElementById("CentigradeIN").value));

To get the value from the input box to the variable cTemp, the program uses a standard statement **document.getElementById** to find the value in an input box identified as **CentigradeID**

See (var cTemp=(**document.getElementById**("**CentigradeIN**").value);

Line 2 of the function is also used in programs 1 and 2 to perform the conversion calculation to Fahrenheit and to store the result in the variable **fTemp**

See (var **fTemp** = ((cTemp \* 9) / 5) + 32;)

Line 3 of the function :

* gets the value stored in **fTemp** andoncatenates fTemp into a string

See document.getElementById("result").innerHTML=**"Fahrenheight is "+ fTemp;**

* The string is assigned to an id called **result** using a standard statement

See **document.getElementById("result").innerHTML**="Fahrenheight is "+ fTemp;

The function ends with the closing brace **}**

Outside the function, the program then the program places the string into a paragraph (that is between the markers <p> and </p>) with an id of **result**

See <p id="**result**"></p>

**Program 4 Add a Reset Button**

This program adds a Reset Button to Program 3

Reset is a standard facility. It is often used to clear the contents of input boxes and outputs. This often help users when making repeated inputs and outputs.

The program opens with the same on-screen objects as Program 3 plus a RESET button.



The results are displayed in a string with the same code as for Program 3

When the RESET button is clicked, the input box is cleared, but in this case the output string is not cleared. (See Program 5). The program is then ready to accept new input.

This program demonstrates that programs can have several buttons on screen and therefore can execute several functions.

The code is Next page:

**Program 4 (Continued) Code that differs from that in Program 3 is emboldened**

<!DOCTYPE HTML>

<html>

<head>

<meta name=”viewport”content=”width=device,initial-scale=1”>

<title>Program 4</title>

<style></style>

</head>

<body>

**<form id="myForm">**

<fieldset>

The fieldset is enclosed within a form with an id of “myForm”

<label for "Centigrade">Centigrade:</label>

<input type="number" select id = "Centigrade" /><br/>

<input type="button" value="GO" onClick="MyFunction()" />

<input type="button" value="RESET" onClick="reset()" /><br/>

</fieldset>

**</form>**

<script>

function MyFunction()

{

var cTemp=(document.getElementById("Centigrade").value\*1); var fTemp = ((cTemp \* 9) / 5) + 32;

document.getElementById("results").innerHTML="Fahrenheight is "+fTemp;

}

RESET function clears down the contents of a form with an id of myForm

**function reset()**

**{**

**documentGetElementByid("myForm").Reset();**

**var fTemp = 0;**

**document.getElementById("results").innerHTML="Fahrenheight is "+cTemp;**

**}**

</script>

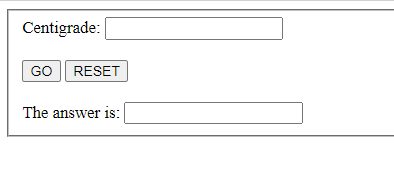
<p id="results"></p> </body> </html>

Program places the results of either the output from myFunction or RESET function

**Program 5 Temperature Conversion within a Form**

This program opens with a screen that looks ‘tidier’ than Program 4. The layout of the on screen objects is improved and the results will be displayed in an output box rather than a statement outside the frame as in Program 4.

Also, all the on-screen objects, including the output box are contained within the frame (i.e. *Form*)



There is very little change to the code from Program 4.

The improved spacing is provided by using <br/><br> at the end of a line

The result of the conversion is shown within a new string contained within the form

<label for "displayValue">The answer is: <label><input type ="textbox" name = "results">

The program is Next Page.

**Program 5 (Continued) Code that differs from that in Program 4 is emboldened**

<!DOCTYPE HTML>

<html>

<head>

<meta name=”viewport”content=”width=device,initial-scale=1”>

<title>Program 5</title>

<style>

</style>

</head>

<body>

<form id="myForm">

<fieldset>

<label for "Centigrade">Centigrade:</label>

<input type="number" select id = "CentigradeIN" **/><br/><br>**

<input type="button" value="GO" onClick="MyFunction()" />

<input type="button" value="RESET" onClick="reset()" **/><br/><br>**

**<label for "displayValue">The answer is: <label><input type ="textbox" name = "results">**

</fieldset>

</form>

<script>

function MyFunction()

{

var cTemp=(document.getElementById("CentigradeIN").value);

var fTemp = ((cTemp \* 9) / 5) + 32;

document.getElementsByName('results')[0].value= fTemp;

}

function reset()

{

documentGetElementByid("myForm").Reset();

}

</script></body></html>

**Program 6 takes Program 5 further. (IN PREPARATION)**

This program adds dropdown lists and IF statements

**Program 7 adds CSS code** .

**Example comprising HTML, CSS and JavaScript based on Example 5 Converting Centigrade input to Fahrenheit output**

**(NOTE THIS SECTION IS UNDER REVIEW)**

The HTML ‘framework’ is shown in red; the CSS statements are shown in green; the JavaScript statements are shown in blue.

Points to note

* In Example 1, the CSS statements are located in the <head> section in between <style> and </> style tags, This is known as in-line CSS.(Compare with Example 2 below)
* The CSS styling in this example is trivial i.e. simply changing a few colours, fonts and font sizes. CSS has a much wider repertoire of style attributes, such shapes, layout, margins, padding, and much more. CSS also facilitates *responsive design* i.e. automatically synchronising an App’s output to ‘fit’ the screen dimensions of the end-user’s device (iPad, tablet, pc, phone etc). You can see many examples of where *responsive design* has been applied when you compare the output from a webpage as displayed on your pc versus as displayed your smartphone.
* In this example, the objects (labels, text boxes and buttons) that are coded in JavaScript, I have assigned an id. For example, the label and the input box have an id of **A**, which, I’ve highlighted in green. The buttons carry an id of **Buttons**. The textbox for the output has an id of **Outbox**. In the literature, these ids are broadly known as *Selectors*.
* In the CSS statements, there is a set of style attributes (e.g. colour, font size etc) that correspond to the *Selectors* quoted in the JavaScript. Note that the selectors are prefixed with # in the CSS section, e.g. **#A**, but **A** in the JavaScript
* Note that in this example, there is a style set for **#scaleout**, but there is no corresponding object identified in the JavaScript. More on why, later.
* As an exercise, change the colours, font sizes etc and change a selector. For example, observe the results when you change #A to #B in the CSS section. Change it back, but then change the selector of the inbox from #A to #Z. The change it back. You could also reverse the colour scheme for the buttons. For example, the caption on the GO button is red on a yellow background. Keep to that scheme, but reverse it for the Reset button, i.e. make the caption yellow on a red background. How would you do that?

**Example 1 In-Line CSS (NOTE THIS SECTION IS UNDER REVIEW)**

DOCTYPE HTML>

<html>

<head>

<meta name=”viewport” content=”width=device, initial-scale=1”>

<title>Example 6 CSS</title>

<style>

#A

{ font-family:verdana;

font-size:20px;

color:red;

background-color:yellow;

}

#Buttons

{ font-family:verdana;

font-size:12px;

color:red;

background-color:lightgreen;

}

#ScaleOut

{ font-family:verdana;

font-size:20px;

color:red;

background-color:yellow;

}

#OutBox

{ font-family:verdana;

font-size:40px;

color:red;

background-color:gray;

}

</style>

</head>

<body>

<form id="myForm">

<fieldset>

<label for "**A**">Centigrade: </label>

<input type="number" select id = "**A**" ><br/><br>

<input type="button" value="GO" onClick="MyFunction()" select id = "**Buttons**" />

<input type="button" class = "floatright" value="RESET" onClick="reset()" select id = "**Buttons**" /><br/><br>

<label for "display">The answer is: <label><input type ="textbox" name = "display" select id = "**OutBox**" >

</fieldset>

</form>

<script>

function MyFunction()

{

var cTemp=(document.getElementById("A").value\*1); // temperature in Celsius

var fTemp = ((cTemp \* 9) / 5) + 32; // temperature in Fahrenheit

document.getElementsByName('display')[0].value= fTemp; // temperature in Fahrenheit

}

function reset()

{

documentGetElementByid("myForm").Reset();

}

</script>

</body>

</html>

**Example 2 Using an External CSS file ( NOTE THIS SECTION IS UNDER REVIEW)**

This example has identical HTML and JavaScript as in Example 1 above.

The output is identical to that produced by Example 1.

The difference is in how CSS statements are invoked. In this case, the CSS statements are stored in a separate (external) file to that of the HTML file (that contains the HTML and JavaScript.

Essentials:

* The CSS file must be saved with a file extension of CSS and located in the same folder as the .HTML file.
* For Example 2, the CSS statements have been cut from the <style> </style> section in Example 1 and then saved in a file called StyleSheetforExample6CSS.css (The statements are saved as is)
* In place of the <style> </style> section, a standard HTML statement is used to refer to the css stylesheet , i.e.

<link href = StyleSheetforExample6CSS.css rel = "stylesheet">

**Example 2 (Extract)**

<!DOCTYPE HTML>

<html>

<head>

<meta name=”viewport” content=”width=device, initial-scale=1”>

<title>Example 6 CSSwithCSSfile</title>

<link href = StyleSheetforExample6CSS.css rel = "stylesheet">

Replace the <style> </style> block with a link to the .css file, e.g.

(StyleSheetExample6CSS.css)

that contains the CSS statements

</head>

<body>

**Appendix 4 (IN PREPARATION)**

**Appendix 5 (IN PREPARATION)**

**Appendix 6 (IN PREPARATION)**

**Appendix 7 (IN PREPARATION)**

**Appendix 8 (IN PREPARATION)**