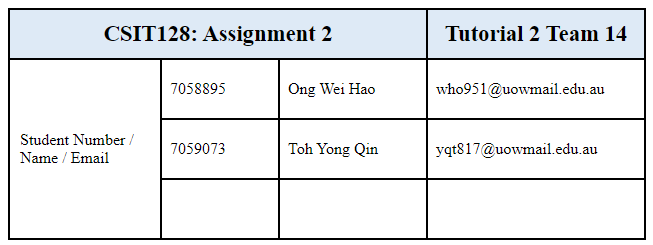
CSIT128: Assignment 2

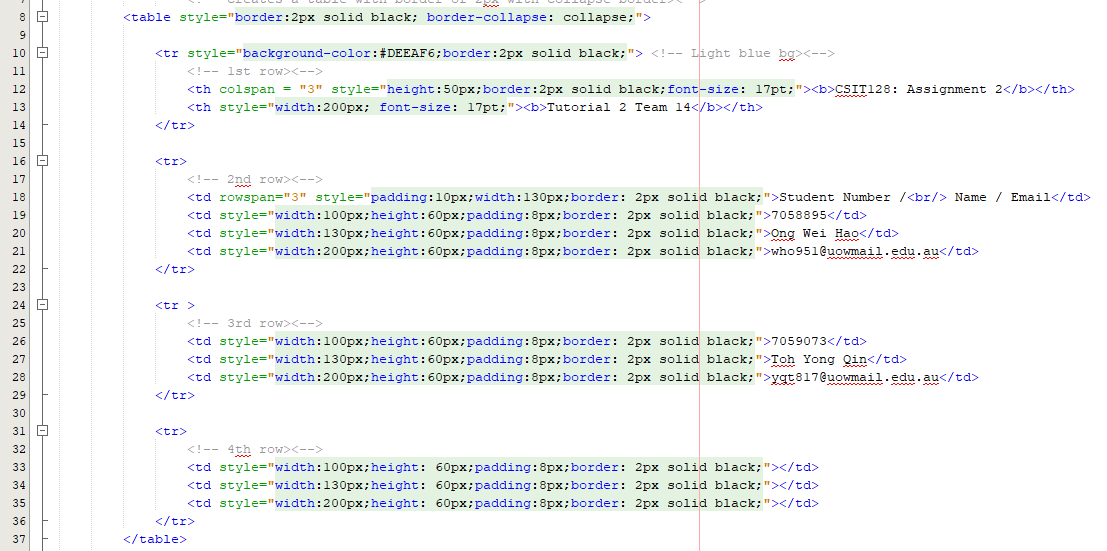
Tutorial class and group: T214

|  |  |  |
| --- | --- | --- |
| Student Number: | 7058895 | 7059073 |
| Name: | Ong Wei Hao | Toh Yong Qin |
| Email: | who951@uowmail.edu.au | yqt817@uowmail.edu.au |

Part 1:

Image:





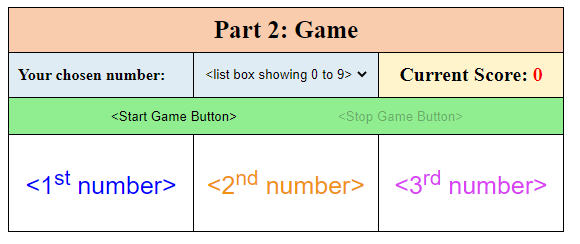
Explanation:

For <table> tag, on default, all cells of the table are separated meaning there is a gap between all cells. Setting border-collapse as collapse ensures no gaps in between each cell and also 2px solid black as the table border.

Moving on to the <tr> tag in the 1st row, our team has decided to set solid black 2px again individually for the cells because the previous one is solely for the table outline, whereas this only applies for the 1st row. Our team has also changed the background color to light blue for the first row similarly as what was depicted in the question. Next, for the first header with inner html "CSIT128:Assignment 2", as the cell covers 3 columns, setting colspan as 3 merges 3 columns into 1 column. <b> changes the wordings to bold and font-size to increase the font size, similarly for the 2nd header as well.

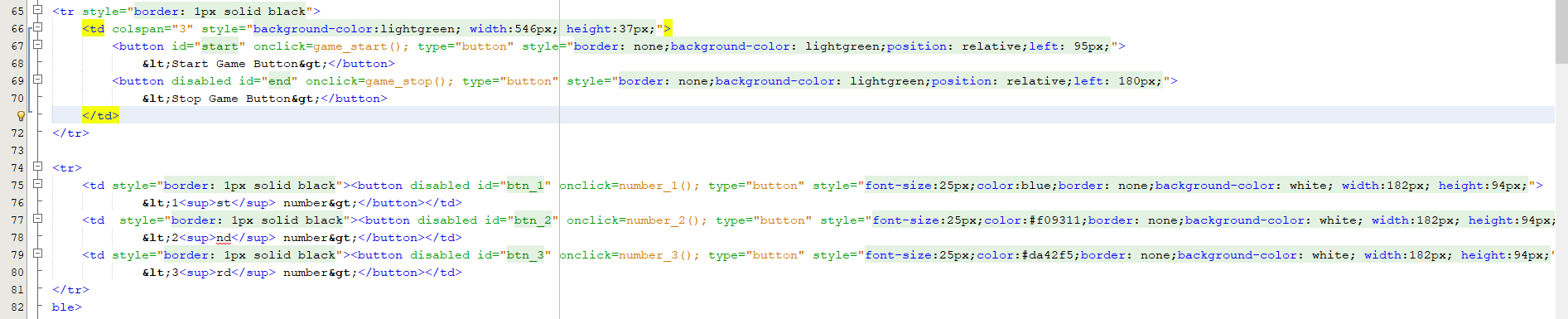
As for the 2nd row <td> tag, it consists of 3 rows. Setting rowspan as 3 combines 3 rows of the same column together. <br/> means the moving to the next line. Moving on to the last few cells, height and width are set as the same for the corresponding column to ensure that for cases where there is empty data in the cells, they will not be compressed down. Thus setting minimum width and height ensures consistency amongst the cells.

Part2:

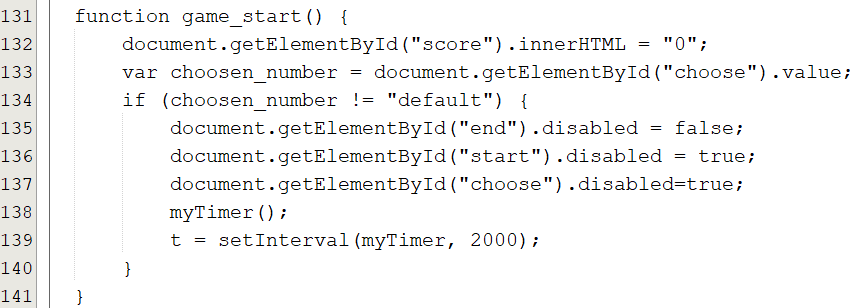


Code:

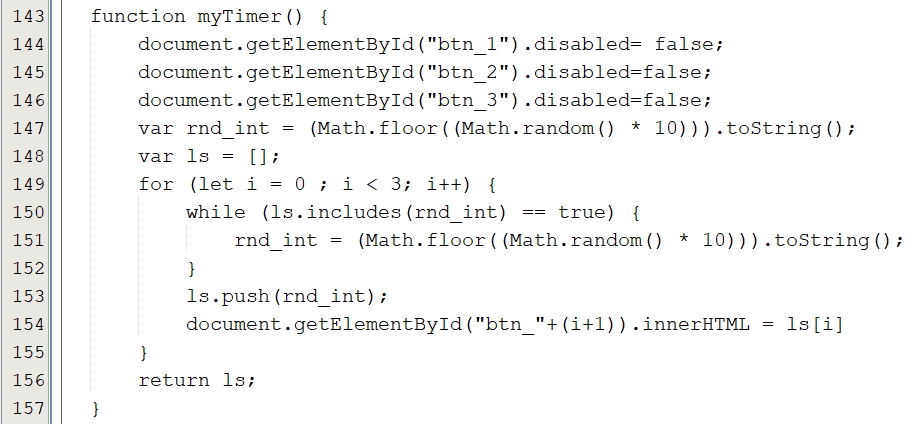




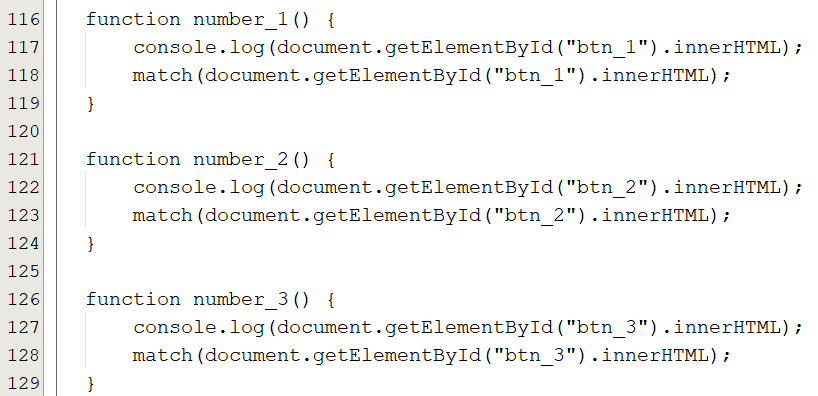
For part 2, the game consists of 4 rows, therefore we have 4 <tr>. For the 1st <tr>, since one of the rows has 3 columns, we have to set the colspan as 3, to combine as 1 column. The next <tr>, we have three columns. The first column with the text "Your Chosen number", the next column with the select list of values consist of 0 to 9. The default is set as "list box showing 0 to 9", but the user will have to choose the values from 0 to 9 otherwise the game can never start. Next column we have the score. The score is set initially at 0 but will increment by 4 points if the user clicks on the correct number and deducts by 3 if it's wrong. The value of score is set inside the <span> tag, with id="score". 3rd row I combine 3 rows into 1. I used relative position left to position both start and end buttons. Next row, I created 3 buttons where each button is in one cell. On default the buttons are set as disabled and enabled upon starting the game.



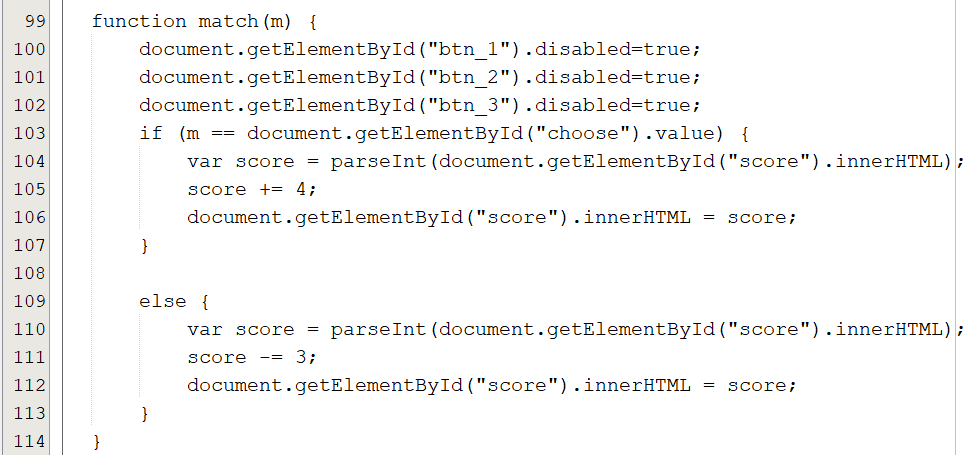
First, the game\_start() function. I set the <span> whose id="score" text as "0" to restart the score where the game might not be the first. Next, I get the value of the select list. If it's default, the game will not start. Else, I enable the end button for the user to end the game, disable the start button so the user will not execute this function many times at once, and disable the select list whose id="choose". Next I run the myTimer() function and set the execution of the myTimer() function in an interval of 2 seconds and stored it as variable t which is a global variable to be used for ending the timer at the end game button.



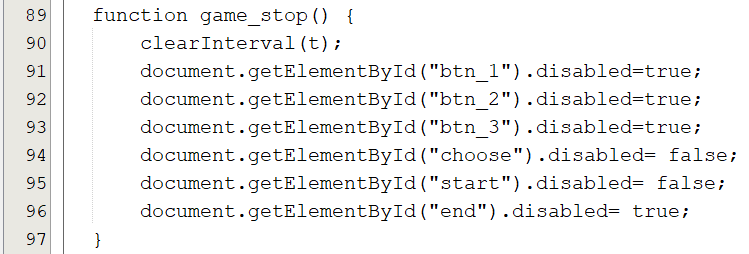
Moving on to myTimer() function, I enable the 3 buttons which the user is supposed to click at the last row. I created an array to store 3 integer objects ranging from 0 to 9 and conditioned it where no same integers are allowed to be inserted inside the array and replaced the innerhtml of each button in a for loop of 3 loops.



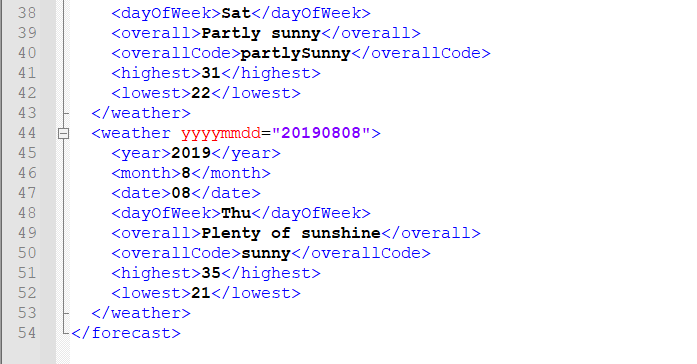
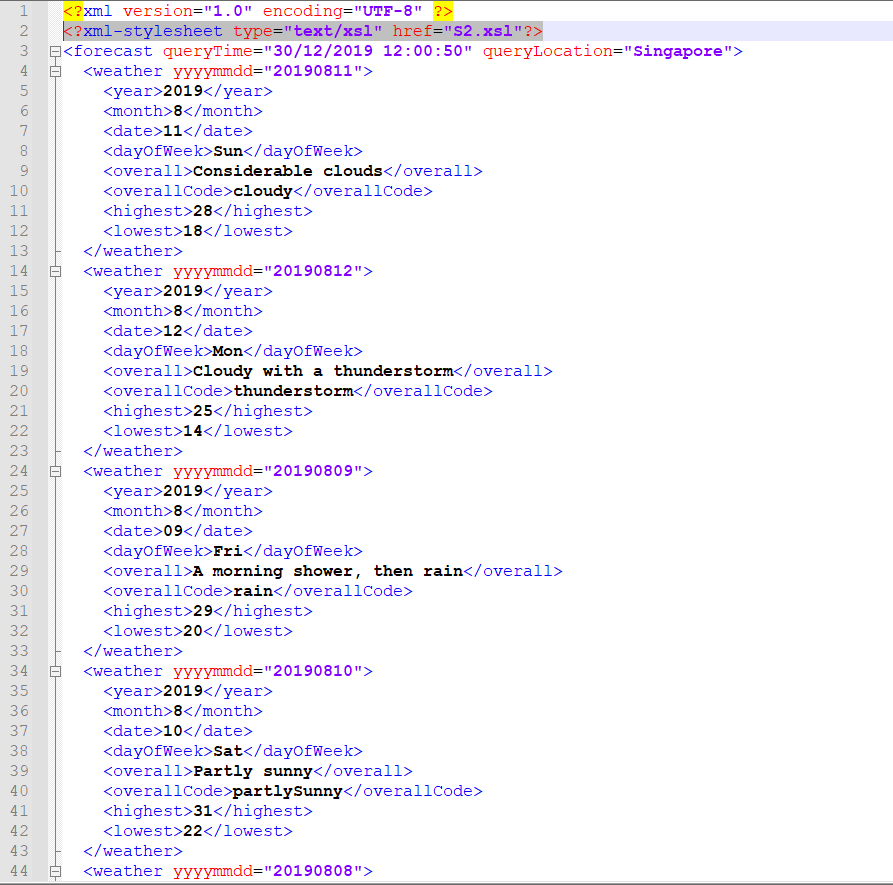
Next for the 3 buttons at the last row, I created 3 functions for the buttons individually. I created a function called match(m) which is stored inside each of the 3 functions.



For the match(m) function, I take in m as the innerhtml of the button clicked. If the innerhtml corresponds to the value of the select list whose id="choose", we update the score by 4 points else deduct by 3. Also, we disable the 3 buttons to prevent the user from over clicking and getting additional or losing points within the 2 second timeframe.



Last but not least, the game\_stop() function. Using the global variable t as we have stated at the start of the <script> tag, we use the clearInterval(t) function to end the setInterval() function.

Part 3 XML

Explanation:

For XML file, a line of code is added in, <?xml-stylesheet type="text/xsl" href="S2.xsl"?>. This is to link it to the S2.xsl file. The rest of the code in XML remains untouched.

Part 3 XSD



Explanation:

For xsd file, xsd:element name=”forecast” is because the root element of A2.xml is forecast. And xsd:element name=”forecast” is referencing the root element. There is a xsd:complexType as in this element, it contains another element, which in this case, is element name weather. It is then followed by <xsd:sequence> as there is a sequence of elements named “weather”. After which will be the xsd:element name=”weather” which is below the root element “forecast”. In the weather element, there is the attribute minOccurs and maxOccur. minOccur = 0 and maxOccur = “unbounded” means there is no limit to the number of times elements occur. It is then followed by a complexType and sequence again as there is another element inside element name=”weather” and there is a sequence again which is the elements that is inside name=”weather”, which is <xsd:element name="year" type="xsd:integer"/> ,<xsd:element name="month" type="xsd:integer"/>,<xsd:element name="date" type="xsd:integer"/>, <xsd:element name="dayOfWeek" type="xsd:string"/>, <xsd:element name="overall" type="xsd:string"/>, <xsd:element name="overallCode" type="xsd:string"/>, <xsd:element name="highest" type="xsd:integer"/>, <xsd:element name="lowest" type="xsd:integer"/>.

type=”xsd:integer” represents that the value is integer

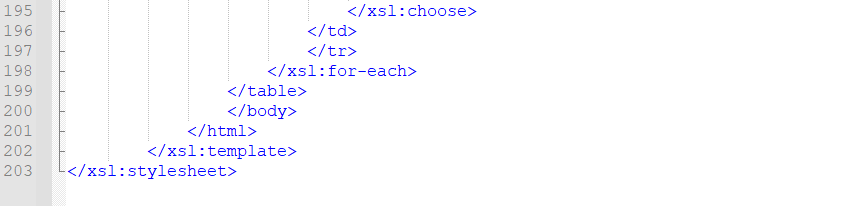
type=”xsd:string” represents that the value is a string

<xsd:attribute name=”yyyymmdd” type=”xsd:integer”/> comes from the attribute that is in element name=”weather”

<xsd:attribute name=”queryTime” type = xsd:string” /> and <xsd:attribute name="queryLocation" type=”xsd:string”/> represents the attribute of forecast.

All attributes have to be placed before closing of the complexType.

Part 3 XSL



Explanation:

For <xsl:template match=”/forecast”>, it links back to the A2.xml root element, which is forecast.

In style, th, table header is set to have background orange and width of 100px. In body, xsl:value-of select=”@queryLocation” is to retrieve the value of attribute queryLocation in the XML and xsl:value-of select=”@queryTime” is to retrieve the value of attribute queryTime in the XML. Table is set to align:center, and first th Is set to have background orange and width of 70px.

For the xsl:for-each select=”weather” is to select each of the sub element that is under element weather and I used <xsl:sort select=”date” to sort them by date, and used style=”text-align:center” in <tr> to align texts in the center. Also, I used <xsl:choose> <xsl:when test=”month=’1’”> to check if month in xml is 1, and if it is 1, Jan will be displayed. Followed by 2 all the way to 12, displaying their respective month in short form(Jan,Feb….) .

For td, I styled it accordingly and put in <xsl:value-of select=”date” which is the date. <xsl:choose> <xsl:when test=”date=’12’ and dayOfWeek=’Mon’”> means when date is 12, dayOfWeek is Mon, I input the value of lowest, highest, input the image and below the image, put the value of overall text. late, lowest, highest, overallCode, overall are the names of the element that is under element weather. Same goes to the rest of the table data, I insert them accordingly to where it is supposed to be. After inserting, </xsl:for-each> is to close the tag and </table> to close the table. Followed by </xsl:template> and </xsl:stylesheet> to close the template stylesheet.