(**function** updateScore(){

setTimeout(**function**()

$.ajax({   
 url: "weather.json",   
 type: "GET",   
 dataType: "json",   
 success: **function**(response){

**let** sTxt = "";

$("#forcast").html("");

sTxt +="<tr><th id='index'></th><th>City</th><th>Weather condition</th><th>Temperature</th><th>Wind speed</th><th>Wind direction</th><th>Wind chill factor</th><th></th></tr>"; $.each(response.weather, **function**(index){  
sTxt += "<tr><td>" +  
 response.weather[index].cityID

+ "</td><td>" +  
 response.weather[index].cityName

+ "</td><td>" +  
 response.weather[index].currerntCond

+ "</td><td>" +  
 response.weather[index].temperature.amount + response.weather[index].temperature.scale

+ "</td><td>" +  
 response.weather[index].windDirection

+ "</td><td>" +  
 response.weather[index].windSpeed.amount + response.weather[index].windSpeed.scale + "</td><td>" ;

**if** (response.weather[index].hasOwnProperty('windChilFactor')){

//Self-executing, recursive function, that runs first when the page is loaded, by executing itself.

// A setTimeout function sets a timeframe after which its (anonymous) function parameter will run.

//A jQuery function then issues an (asynchronous) http request to get data in json format from a source. It takes an object, as parameter, with five properties, including two callback functions: one that

should execute when the request is successful, and the data has been received and another function to execute in case of an error. The other properties specify the source url, the type of request, GET or POST, and the format for the received data. (The source has to provide/offer data in that format.)

// sTxt, a ‘block exposed’ variable that will be used to build up and to accumulate the html to be inserted into the html DOM and to be displayed to the user, later.

//Clears first the tag with the id ‘forcast’ in the html DOM that may still contain a value from the previous load.

//Builds the column headers for the table.

//The purpose of this function is to enable traverses over the array within the response object. For each item/object in the array (in effect another city’s data) a function is provided that takes the index of the current item in the array. With this index the function extracts the values from the individual name-value items, such as cityID or Cityname, from within the ‘weather’ array. These can also be extracted from name-value items within further nested objects such as temperature or amount.

The values obtained are then surrounded by html cells or/and row tags. The purpose of this is to incrementally build up the table by constructing one row for each city.

//Checks if the ‘wind chill factor’ pair exists as it only applies for temperatures between -50 and 50 F and wind speeds above 3 mph.

sTxt += response.weather[index].windChilFactor.amount + response.weather[index].windChilFactor.scale + "</td><td>" ;  
 }

**else** {  
 sTxt += "N/A</td><td>";  
 }  
   
 sTxt += "<img src=\"weather\_icons/" + response.weather[index].iconName + "\"\\>" + "</td></tr>";  
   
   
   
 });  
 $("#forcast").append(sTxt);   
 updateScore();  
 },

error: **function**(){  
 $("#info").html("<p>An error has occurred</p>");  
   
 }  
 });  
 } ,250);  
   
})();

//Builds the weather icon img tag and src attribute.

//We now target the html tag that contains the id ‘forcast’ and we append to it the finished sTxt we built. Right after that we call the updateScore() function again, recursively. Note, how this is a demon function, that is, it never stops calling itself and therefore so long as the page is loaded keeps calling itself.

//In case of an error, this function picks up the div containing the id ‘info’. It then inserts this p message into the html node to be displayed to the user.

//The time frame, 250ms, after which the function given to setTimout should execute.

//The bit that does the self-executing trick, the first time that the page is loaded.