Techniques

* LocalStorage and Database workaround
* Functions
* Random number generation
* Timing Events
* Layout using HTML tables
* CSS stylesheets

Additional information is available in the appendices:

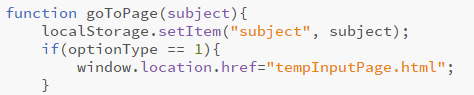
* Evidence of development
* Bibliography

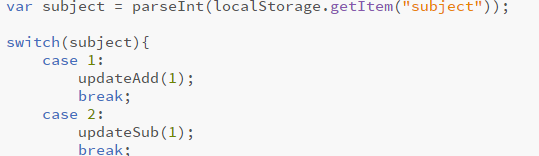
HTML and JavaScript, while being compatible on many platforms and very good performing web related tasks, turns out to have many disadvantages that need to be overcome when it is in the form of an offline application. The fact that it runs in a browser creates limitations. The most major one is the lack of databases that can be used since a server side script cannot be used.

**LocalStorage**

“LocalStorage” is used in the program in order to store information and data long term, as well as passing information between different pages. Passing information between different pages is required since no information can be passed to the next page when changing the current page displayed in the window. Usually the information is passed to the server which processes the information and displays the page correctly back on the client side. However in this case there is no server. In order to solve this, the LocalStorage function is used. The local storage function is client side and stores long term data in the hard drive that persists through window location switching.







One of the examples I have used for this is in the home page. For example, when the ‘addition’ button is clicked, a value is stored as a string using local storage and then the page window is switched. The new page, in this case the quiz page, uses a switch statement to determine the corresponding action, i.e. display addition questions.



The application also needs to store data despite closing the app for long term storage for values such as score histories. Local storage conveniently solves that problem in that it provides long term storage locally. However, a huge limitation is that it can only store strings, not objects that aid in database creation such as arrays. Thus values need to be converted into their correct data type each time it is retrieved. In order to solve this, the data is stored by delineating a data member with commas and other symbols in a format similar to a comma separated file. In order to read the data, a function reads the string and puts it in an appropriate format at the receiving end.

An example of this is the “convertIntoArray()” function for creating inputs for the graph generation. The “string.split(,)” creates an array with each element delineated by a ‘,’. The format of the string and the array created from it is “date, score, date, score…” In order to interpret the data, a ‘for’ loop is used, but instead of incrementing by one it increments by two. This means that if the initial value for ‘i’ is set to be 0, then it will only retrieve the dates, while if the initial value for ‘i’ is set to 1 then only the score will be retrieved. “array.push()” puts a value at the end of the array.

**JavaScript Function**

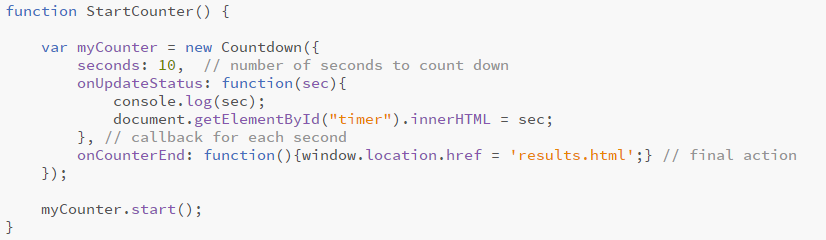
Functions are also used extensively in the application in order to reduce code clutter and make code modification and maintenance relatively easy. Most reusable functions are put in the “common.js” file.





**JavaScript timing events**

JavaScript timing events are used in order to create a countdown timer for the “Quick Fire” Gamemode.





In order to start the timer, the “StartCounter()” function is called, creating a new Countdown object.

The countdown object decrements the counter by 1 every second (1000 milliseconds) using the setInterval() function. The parameters passed are the action taken and the time waited respectively.

When the counter reaches 0, the window location is switched to the results page.

**Random Number Generation**

A key component of the game is generating random numbers for the quiz questions.

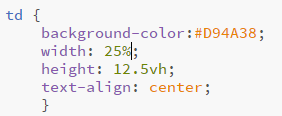


The since the numbers being quizzed on should be integers, “Math.floor” is used. In order to generate questions with differing difficultly, the parameter “difficulty” is passed and a switch statement determines what ranges the values of the numbers can take on. The multiplication determines the maximum value since “Math.random()” goes from 0 to 1. The addition part determines the minimum value after the maximum is determined and also increases the maximum.

**Layout**

For layout, HTML Tables are used. This is because it is easy to create a regular grid layout suitable for an app. In order to fit screens of different sizes, the sizes of each grid was defined using percentage values. ‘vh’ is also used to set the sizes relative to the vertical height.

|  |  |
| --- | --- |
|  |  |



**CSS stylesheets**

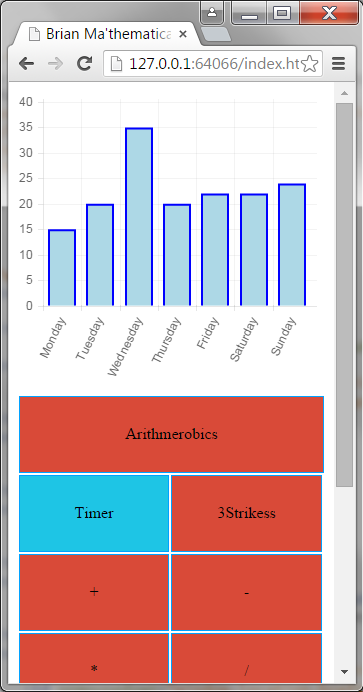
A CSS stylesheet is used to create a unified aesthetic for the app. This saves the amount of work needed for individually defining styles, as well as reducing code clutter, making maintenance easier.





**Chart Creation using Chart.js library**

In order to create a graph for the statistics, the Chart.js library is used. Parameters needed for the creation of a graph are “label” and “dataset”. The “dataset” parameter includes options for graph generation as well as the data for the graph.





**Section D Plan**

* Show home page
* Scroll up and down
* Click on QuickFire
* Click on addition
* Get question right
* Get question wrong
* Showcase decimal input
* Get as many right until timer runs out
* Show results page
* Show graph changes
* Click on Three strikes
* Get question right
* Get question wrong until all strikes
* Show results page
* Show try again button
* Get all wrong to get to results page
* Click on home
* Show custom game choices with 3 strikes
* Get all wrong
* Show results
* Show tutorial pages
* Exit back to home
* Show about page
* Exit back to home
* End of Video