Rich-Web Lab 3 Exercises

1. Explain what is meant by the stream abstraction. What is the relationship between streams and the observer pattern? What are streams useful for modeling and when might you use them in Rich Web development?
   - Stream Abstraction is a non-mutable collection of functions that are applied in a specific order of data however it is not a collection where you would store data elements. The observer pattern is a software design pattern where an object aka the subject keeps a list of its dependents called the observers and on the other hand the stream is basically the observable being observed in this context. Streams are useful in viewing data streams or sequences of events in time and uses them as a model for the central input and output objects in rich web development. They are also useful for dataflow programming, reactive programming, and also distributed data processing developing web pages.

2. Assume that you are building an interface to an API in your Rich Web App. Describe in detail how you could use the RxJS library to handle asynchronous network responses to API requests. In your opinion, what are the benefits to using a streams library for networking over, say, promises? And what do you think are the downsides?
   - The RxJS library can be used as an implementation of the Observable type which is needed until the type becomes part of the programming language until browsers can support it. The RxJs library provides utility functions for creating and working with observables and can also handle asynchronous network responses quite well. With an RxJS library we can request multiple things at the same time to a rich web application. Requests are met, in our web application, on a first come first served basis and after multiple requests it provides a response to all the requests in its personal contained thread. The benefits of using a stream library are that a stream library can

encourage writing code in a functional style rather than in an imperative style which mitigates a callback problem and overall improves the code cohesion especially for networking more than promises do but they share the ability to transmit any set of data with the strict type checking by the compiler. However, a downside with streams is that you can't guarantee that the stream will send any value events before it's finished and this is especially true for networking as the client code will expect to receive the network response only once but instead will receive multiple value events if the request streams the data so that is a big downside.

3. Consider three asynchronous tasks, A,B & C. What are the consequences of these functions sharing global state? What is a good practice to alleviate any problems associated with this?

- If the three functions share a global state, then any changes to the state made by any one of the functions will be seen by the others. This could result in race circumstances, in which two or more functions attempt to access and modify the shared state concurrently, producing unpredictable outcomes. Use of locks is one technique to solve this issue. Before gaining access to and making changes to the shared state, each function can obtain a lock. As a result, race situations are avoided because only one function is ever able to modify the state at once. A different solution to this issue is to completely prevent sharing state across the functions. Each function is able to keep its own private state and only exchange information with other functions through message passing. By doing this, race conditions are prevented and each function is guaranteed to have total control over its own state. Finally, the functions might make use of a shared data structure like a concurrent map that is intended to be used by numerous concurrent threads. Race conditions would be avoided because to the synchronization techniques provided by this data structure, which would limit the number of concurrent state modifications to one.