## Lecture Review Questions

1. Explain what is meant by the stream abstraction. What is the relationship between streams and the observer pattern?  What are streams useful for modelling and when might you use them in Rich Web development?

*Explain what is meant by the stream abstraction*

Streams are collections of data just like arrays or strings. The difference is that streams might not be available all at once, and they don’t have to fit in memory. This makes streams powerful when working with large amounts of data, or data that’s coming from an external source one chunk at a time. This is where abstraction comes in as the programmer doesn’t have to know certain details for certain, like the size of the stream etc.

*What is the relationship between streams and the observer pattern?*

Streams implement the observer pattern meaning they notify the observer if or when a change occurs such as there is no more data available or that an error has occurred.

*What are streams useful for modelling and when might you use them in Rich Web development?*

Streams allow the application architecture to reduce to a stream processing problem operating on a merged set of one or more data streams. This provides a unified abstraction of everything. For example, having mouse clicks, keyboard input, network responses, timers and the DOM state changes can all be processed within the same logical structure using the same semantics.

1. 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?

*How you could use the RxJS library to handle asynchronous network responses to API requests*

You can create observable streams to handle API requests. In Rx, we have stream functions such as map, filter, scan, merge, combineLatest, startWith, and many more to control the flow of an event-driven program. This toolset of functions gives you more power in less code. With streams it will allow you to handle the response of an API as a stream, allowing you to access data quicker. You can then use that stream and break it down into smaller streams using functions such as map and filter.

*What are the benefits to using a streams library for networking over, say, promises?*

Streams allow for faster responses for things such as webapps that are highly interactive with a multitude of UI events related to data events. As apps evolve to be more real time, like modifying a single form field can automatically trigger a save to the backend which links to some contents that can then be reflected in real time to other connected users.

With promises you can only handle one asynchronous operation with each promise. That's fine for sending HTTP requests and reacting to responses for example. It's not really a great solution if you want to handle asynchronous operations which don't end after one "value".  Promise is simply an Observable with one single emitted value. Rx streams go beyond promises by allowing many returned values.

*And what do you think are the downsides?*

One potential downside of using Rxjs streams is that you might end up wrapping all your data in Observables. This might lead to making it more complex to support client-side code, hot module reloading or time travel debugging.