# Rich Web - Lab 3

## Question 1

Streams abstraction is a way in which we can model asynchronous data sources. A stream is a power technique when processing data when you either don’t know its potential size and/or you don’t know when it will arrive into your application. Processing of streams generally must be done in time-separated chunks in sequence or concurrently, depending on the application. A stream is understood to be a time-ordered list of items which you read and transform into other streams and values in some application-specific order.

Streams implement the observer pattern where data is realised through the use of the subscribe operation. These operators can be attached to multiple different web features such as mouse clicks or keyboard input. Streams are useful for modelling data which may or may not be present now or may or may not arrive in the future. Streams can be used to check for errors in the data being passed in or to check when all the data has finished being passed in.

## Question 2

When building an interface to an API in a rich web application you can use the RXJS library to handle the HTTP requests and responses. The RXJS makes it easier to get HTTP responses and process the data retrieved. Once you get a response you can use the error function to check whether the data you are getting actually exists and you can also use the complete function to easily and efficiently end the stream when all the data has been received. Using the error function give you the ability to easily check if the API you are trying to reach actually exists or not. If the API doesn’t exist, then it will be caught, and the specified error message will show.

The benefits of using streams library for networking over promises, in my opinion, are the following:

1. Subscribe is very powerful because it can be used with totally different types of operations and we don’t have to worry about having to change it every time we want to work with a different operation.   
   E.g. Rx.Observable.range(1, 10) .subscribe(e => console.log(e));  
    Rx.Observable.fromEvent(document, 'mousemove') .subscribe(e => console.log(e.clientX));  
   Whereas with promises you have to change it every time you want to do something different.
2. With observables, other objects a can register and listen to them. If the Observable object changes, it doesn’t necessarily have to require a re-write of anything else in the system. The other class doesn’t have to change at all in order to interact with the Observable. Multiple objects can register and observe the Observable object.

The main downside to streams is that observables cannot be chained. Observables return a subscription to the observable object. Promises can be chained together. They return another promise.