Rich Web Lab 5+6

Part 3
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?

- Streams are an abstraction used to model asynchronous data sources.
- Streams implement the *observer pattern* where data is realised using the subscribe operation.
- A stream is a powerful 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
- One possible solution to the synchronisation problem is to model all application state as streams, providing a unified abstraction of everything
- Using streams everywhere allows the application architecture to reduce to a steam processing problem operating on a merged set of one or more data streams

Part 4
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?

```
const request$ = Observable.just('https://api.github.com/users');

const response$ = request$
  .flatMap(requestUrl =>
    Observable.fromPromise(fetch(requestUrl))
  );

response$.subscribe(response => {
  // render `response` to the DOM however you wish
});
```

- We can use the just operator to create a stream based on the string URL for the end point that we are going to get the data from. This request stream will request a series of request stream based on the url requested.
- It then flat mapped into our response stream. Our request stream will produce a stream of stream
- We then can use the built in fetch operator which will return a promise and from that promise we can create another stream using the observable fromPromise operation.
- Once we have out responses, we can subscribe to those responses and for each response that arrive we can render it into the DOM.

- With a promise you have a value or an exception, and with a stream, you have a list of values or an exception.
- A Promise handles a single event when an async operation completes or fails.
- A stream allows to pass zero or more events where the callback is called for each event.
- Streams are better because it provides the features of Promise and more. With Observable it doesn't matter if you want to handle 0, 1, or multiple events. You can utilize the same API in each case.
- Stream is cancelable wheres as promises are not
- Stream uses maps and forEach