Angular 6 Training Course

Exercise G-observable

- **Reactive programming** is an async programming paradigm that organises code around data streams.
- **Observables** are objects which create streams, and subscribe to the data they supply.
- Observables may form part of **ES7**.
- **rxjs** (Reactive Extensions for JavaScript) is a library that provides an implementation of observables for JS.
- Observables are widely used in Angular. For example, monitoring the changing state of a form, or the data from a connection to a **Firebase** database is implemented using Observables.
- You do not have to write your own code using Observables, but an understanding of Reactive programming will be useful.

Streams

- **Streams** are a sequence of values over time.
- An example stream is a sequence of numbers which double in value, generated once a second.

```
[ 1,2,4,8,16 ]
```

- Other examples:
    - A stream of coordinates from **mouse clicks**.
    - A stream of **keypresses**.
    - A stream that captures the changing state of a **form**.
    - A stream of data from an open real-time **websocket** connection between a client and a server database.
- An observable will start emitting a stream of values, once something **subscribes** to it.

Observables in plain JS

- Once we load the rxjs library, we can start creating streams.

```
<script src="rx.all.js"></script>
```

- This code defines a simple stream of numbers that will be generated once per second.

```
let sequence = Rx.Observable.interval(1000);
// [ 0,1 .. ]
```

- The stream will only start emitting values (become hot) once we **subscribe** to it.

```
let sequence = Rx.Observable.interval(1000);
sequence.subscribe(v => console.log(v));
```

- The stream is generated faster if we reduce the interval.

```
let fastSeq = Rx.Observable.interval(50);
fastSeq.subscribe(v => console.log(v));
```

- The example streams above are infinite in length.
- We can limit stream length by chaining the **take** method.

```
let alphabet = Rx.Observable
    .interval(50)
    .take(26)
    .subscribe( l => console.log( l ))
```

- This generates a stream of 26 numbers from 0 to 25.
- To turn this into the letters of the **alphabet**, we can add a **map** function to the chain.
- This built-in function converts numbers to Unicode values:

```
String.fromCharCode( 65 ); // returns "A"
```

- This will generate the letters of the alphabet:

```
let alphabet = Rx.Observable
.interval(50)
.take(26)
.map( l => String.fromCharCode( l+65 ))
.subscribe( l => console.log( l ))
```

- This example generates random sequences of the four DNA base values

```
let fourBases = [ "adenine", "guanine", "cytosine", "thymine" ]

let dnaSequence = Rx.Observable
.interval(100)
.map( v => fourBases[Math.floor(Math.random() *
```

```
  fourBases.length)])
  .subscribe(value => draw(value,value));
```

- This example picks up keys as the users types into a text input box:

```
<input type="text" id="city">

let city = document.getElementById("city");

let letters = Rx.Observable
.fromEvent( city , 'keypress' )
.map( v => v.key )
.map( v => v.toUpperCase())
.subscribe(v => console.log(v))
```

Angular project

- Create a new Angular project that will use Observables.

```
cd desktop
ng new g-observable
cd g-observable
ng serve --open
```

- Import the Observable type from the **RXJS library** into the main component src/app/app.component.ts.

```
import { Observable, interval } from 'rxjs';
```

- Create a property that will contain a stream of numbers.

```
sequence: Observable<number>;
```

- Create a function that creates a stream.

```
    counter : number ;
    createSequence() {
    this.sequence = interval(250);
    this.sequence
        .pipe(take(20),map(n => n * 4))
        .subscribe( n => this.counter = n );
    }
```

- The stream does not become hot/active until we add a subscriber.
- Display this property in the template.

```
<p>{{ counter }}</p>
```

- Call this method from the constructor.

```
constructor() { this.createSequence(); }
```

- A changing stream of digits will be displayed.

Async Pipe

- We can implicitly subscribe to an Observable stream using the Async Pipe.
- Create a property to hold a stream of 26 letters from A to Z.

```
alphabet: Observable<string>;
```

- Write a function that creates the stream.

```
createAlphabet() {

    this.alphabet = interval(250)
    .pipe(take(26),map(n => String.fromCharCode(65 + n)));

}
```

- Call the function from the constructor.

```
this.createAlphabet();
```

- Add this property to the template with an Async Pipe.
- *Note, the createAlphabet method does not need to explicity use a subscribe function.*

```
<p>{{ alphabet | async }}</p>
```