JS II
Callbacks & Promises

Chapters in JS Book

- Chapter 29 (213 - 218)
  - Callbacks
- Chapter 30 (219 - 220)
  - Intervall & Timeouts
- Chapter 42 (263 - 275)
  - Promises

Callbacks

- What we already know:
  - Functions (Chapter 19)
  - Functions are first-class citizens in JS
  - Anonymous functions
  - Arrow Functions
- Callbacks
  - Next 2 examples are from JS book (page 213)

```
function foo(array)
{
    var sum = 0;
    for (var i = 0; i < array.length; i++)
    {
        alert(array[i]);
        sum += array[i];
    }
    return sum; }
```

```
function foo(array, callback)
{
    var sum = 0;
    for (var i = 0; i < array.length; i++)
        {
        callback(array[i]);
        sum += array[i];
        }
    return sum;
    }
```

Timeout & Interval

- Recursion versus Timeout/Interval

Recursion

```
function foo1()
 {
        //do something
        foo1();
}
```

```
function foo2()
 {
        //do something
        setTimeout(foo2,0);
}
```

SetTimeout

setTimeout(code/functionref);

setTimeout(code/funcionref, delay)

setTimeout(functionref, delay, arg1, … argn);

setTimeout returns timeOutID

https://developer.mozilla.org/en-US/docs/Web/API/setTimeout

clearTimeout(timeOutID);

https://developer.mozilla.org/en-US/docs/Web/API/clearTimeout

Interval

setInterval(code/functionref);

setInterval(code/functionref, delay);

setInterval(functionref, delay, arg1, .. argN);

setInterval returns intervalID

https://developer.mozilla.org/en-US/docs/Web/API/setInterval

clearInterval(intervalID);

https://developer.mozilla.org/en-US/docs/Web/API/clearInterval

Promises

- Promises (Chapter 42)
  - Way to “mask” async calls
- Promise is an object that represents the outcome of an asyn call
  - pending
  - Fulfilled
  - Rejected
- Promise changing
  - then
  - catch
  - finally
- Next 2 Examples from book / page 263, 264

```
const promise = new Promise((resolve, reject) => {
// Perform some work (possibly asynchronous)
// ...
        if (/* Work has successfully finished and produced "value" */) {
                resolve(value);
        } else {
                // Something went wrong because of "reason"
                // The reason is traditionally an Error object, although
                // this is not required or enforced.
                let reason = new Error(message); reject(reason);
                // Throwing an error also rejects the promise.
                throw reason;
        }
});
```

```
promise.then(value => {
        // Work has completed successfully,
        // promise has been fulfilled with "value"
        }).catch(reason => {
        // Something went wrong,
        // promise has been rejected with "reason"
});
```

```
<!DOCTYPE html> <html> <body>

<p id="demo"></p>

<script>

function myDisplayer(some) { document.getElementById("demo").innerHTML = some; }

let myPromise = new Promise(function(myResolve, myReject) {

  let x =  Math.floor( Math.random() * 100);

  if (x < 50) {  myResolve("OK"); } else { myReject("Error"); } });


myPromise.then( function(value) {myDisplayer(value);}, function(error) {myDisplayer(error);} );

</script> </body> </html>
```

fetch

- fetch(resource)
- fetch(resource,options)
- Fetch returns a promise

- https://developer.mozilla.org/en-US/docs/Web/API/fetch
- https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch

fetch(‘http://localhost:8080/ralph.txt’)

=> returns promise

fetch(‘http://localhost:8080/ralph.txt’)

```
<script>
 fetch("ralph.txt").then(res => console.log(res));
</script>
```