Framework Training
React
London July 2018
Javascript review

Variables

- **Case sensitive**.
- **Uninitialised** variables are undefined.

```
var year = 2018;
var city = "Seville"
var smoker = false;
var town; // undefined
var project = null;
```

- **Weakly typed**: the type is not defined and can change.
- Typescript allows enforcing of strict types

```
var town = "Sandwell";
town = -45;
```

- Variables are globally scoped if defined outside a function, or object.

```
window.city === city; // true if global variable
```

Using let in ES6.

- Variables declared with let use **block scope**.
- Here variable j only exists within the for loop.

```
for( let j=0; j<10; j++) { console.log(j); }
```

- Variables can only be defined once with the current scope;

```
let city = "Oslo";
let city = "Copenhagen"; // run-time error
```

Equality

- Loose equality returns true if two values are the same but not of the same type.

```
2+2 == "4" ; // true
```

- Strict equality requires two values to be the same value and the same type

```
2+2 === "4"; // false
```

Truthy, falsy

- Javascript uses a loose boolean concept.
- Almost every expression evaluates to true except for a short list of values:

```
undefined, null, "", false, 0, NaN
```

Arrays

- Arrays are zero-indexed lists.
- Typically, the items in the list are of the same type.

```
let capitals = [];
capitals.push( "Paris" );
capitals.push( "Madrid" );

// Move from back to front
let last = capitals.pop();
capitals.unshift( last );

// Move from front to back
let first = capitals.shift();
capitals.push( first )
console.log( capitals );
```

- We can iterate over an array using **forEach**.
- Define a function that displays one capital.

```
function show( city ) { console.log( city ); }
```

- Call that function for every item in the array.

```
capitals.forEach( show );
```

Objects

- Objects define structured data in a self-documenting way.

```
let fred = { age:64, name:"Fred Smith" };
fred.job = "Postman";
fred.holiday = { city:"Paris", year:2017 };
```

```
fred.smoker = false;
delete fred.smoker;
```

- We can create arrays of objects.

```
let people = [ fred,jane ];
console.table( people );
```

- We can iterate over an array using forEach

```
function getAge( p ) { console.log( p.age ); }
people.forEach( getAge );
```

- **JSON** is the string representation of objects.

```
let s = JSON.stringify( fred );
// "{"age":64,"name":"Fred Smith","job":"Postman","holiday":
{"city":"Paris","year":2017}}"

let ob = JSON.parse( s );
```

Copy by reference/value.

- Numbers and strings are primitive values. Assignment will create new independent copies.

```
let cityA = "Lisbon";
let cityB = cityA;
cityB = "Madrid";
console.log( cityA, cityB ); // "Lisbon","Madrid"
```

- Arrays and objects are complex. Assignment will create two pointers to the same value.

```
let personA = { name:"Bert", age:54 };
let personB = Object.assign( {} , personA );

let lottery = [ 4,5,6,7,8 ];
let lotto = Object.assign( [] , lottery );
```

Functions

- Functions define local scope.
- Functions are hoisted to the top of their containing scope.

```
function double( n ) {
```

```
        let result = n*2; // local variable
        return result;
}
```

ES6 arrow functions

- ES6 introduces arrow functions.
- They omit this syntax: function, (), {}, return
- Here double is the function name.
- “n” is the argument passed in.
- “n*2” is the return value.

```
let double = n => n*2 ;
```

- If we pass in 2 arguments, we need additional parentheses.

```
let calcArea = (a,b) => a*b ;
```

- A function with NO arguments needs paretheses

```
let getYear = () => "2016";
```

- To return an object, wrap {} in ()

```
var createCity = ( c,n ) => ( {city:c, nation:n })
```

Functional Javascript

- Functional JS uses forEach, map, filter and reduce to transform arrays of data.
- **forEach** runs a function for each item in an array but does not create a new array.

```
let lotto = [ 4,10,20,40,45 ];

function show( n ) { console.log( n ) }

lotto.forEach( show );
```

- **map** runs a function on each item in an array and creates/returns a new array.

```
function double( n ) { return n*2 }

let newLotto = lotto.map( double );
```

- **filter** runs a boolean function against each item in an array and returns a new filtered array

```
function getBig( n ) { return n > 20 }

let bigLotto = lotto.filter( getBig );
```

- **reduce** applies a function to adjacent pairs in an array and returns a single value.

```
function add( a,b, ) { return a+b }

let total = lotto.reduce( add );
```

- Functional techniques can be combined with ES6 arrow functions.

```
let double = n => n*2;
let newLotto = lotto.map( double );
```

ES6 constants

- ES6 constants cannot be re-assigned.

```
const YEAR = 2018;
YEAR = 2017; // run-time error
```

- Note, the properties of complex constants can be changed.

```
const HOLIDAY = { city:"Paris",year:2014 };
HOLIDAY.year++; // This works and changes year.
```

ES6 Destructuring

- ES6 destructuring allows variables to be created from complex objects.

```
let { city, year } = { city:"Paris",year:2014 };
```

- Variables can be assigned new names.

```
let { city:c, year:y } = { city:"Paris",year:2014 };
console.log( c,y );
```

ES6 classes

- ES6 introduces class syntax.
- Methods do not need to be prefixed with the word function.
- The **constructor** function is called at instantiation.
- Classes can use inheritance.

```
class Rectangle {

    constructor(length, width) {
        console.log( "Rectangle" );
        this.length = length;
        this.width = width;
    }

    getArea() {
        return this.length * this.width;
    }
}

var rect = new Rectangle(6,4);
console.log( rect.getArea());
```

ES6 modules

- ES6 allows us to define modules: separate .JS files which have their own scope.
- Everything with a module is privately scoped unless it is explicitly made visible with the **export** keyword.

```
// utils.js
let halve = n => n/2; // private
export let double = n => n*2; // public
```

- Exported functions and variables can be imported into other files

```
import { double } from "./utils.js" ;
double(2);
```

- This feature is not available in all browsers: **https://caniuse.com/#feat=es6-module**
- To use this code, add the **type="module"** attribute to the script tag:

```
<script type="module" src="utils.js"></script>
```

- An alternative approach uses **Webpack** to bundle and transpile ES6 module code back to standard ES5.

ES6 template strings

- Multiple line strings can be defined using the back-tick character.

```
let city = "Oxford";
let markup = `<section>${ city }</section>`
```