ES6

Introduction

- ECMAScript (ES) is the JS specification
- ES6, finalised in 2015, is now almost universally supported and includes many features that are used widely in frameworks and libs, e.g. React
- ES6-style JS code is sometimes referred to as modern JavaScript

Template Literals

- A template literal is a string that supports embedded expressions, e.g.:

```
var greeting = `Hello ${user}`;
```

- Note the backticks (not quotation marks) and `${}` expression placeholder

- Template literals recognise line breaks and can be tagged with a custom parser

Default Parameters

- A default parameter is one that is assigned something other than undefined if no argument is provided, e.g.:

```
function f(a, b=2) {…}
```

- The default value will NOT be assigned when the argument is one of the other falsey values

- The default value may be composed of the parameters that went before

Block Scope

- Variables declared with `let` and `const` are block scoped, and…
  - are not props of `window`
  - cannot be re-declared
  - are NOT hoisted

- a `const` variable is a constant, i.e. it cannot be reassigned; NB: this does not mean the object referenced cannot be changed - a constant variable != an immutable object

Spread

- The spread operator (`...`) is used to unpack an iterable; imagine a box of n items being upturned onto the floor, e.g.:

```
const info = ['Dave', 'dave@me.com'];
function register(name, email) {}
register(...info);
```

- Spread is commonly used to copy and extend arrays and objects too

Rest

- The rest parameter (`...`) is used to pack items into an iterable; imagine n items being packed into a box, e.g.:

```
function average(...nums) {}
average(42, 103, 77);
```

- Logically the rest parameter must appear last in the function's parameter list

- Unlike `arguments` the rest parameter is a real array

Destructuring

- Destructuring is the assignment of n array elements/object props into n variables in one go
- It is particularly useful for extracting props from an object to avoid duplicating the object ref

```
const {x, y, z} = coords;
```

- The extracted props may be assigned default values and aliases too

Arrow Functions

- An arrow function is an anonymous function with a simplified form, e.g.:

```
const circ = r => 2 * Math.PI * r;
```

- Rules:
  - the function keyword must be omitted
  - the parentheses may be omitted if one param only
  - the => must separate the params from the body
  - the braces may be omitted if one statement only
  - the return keyword and ; must be omitted if no braces

Enhanced Object Props

- ES5: `var obj = { x : x };`
- ES6: `let obj = { x };`

- Computed property names (using the [] notation) is valid when the object is constructed
- Method props are simplified too - the function keyword is no longer required

Generators

- An iterable is an object that can be iterated over, e.g. an array; to be more precise it is an object that implements the iterator pattern

- A generator is a special function that enables the building of custom iterators, e.g.:

```
function* reverse(arr) {
  for (let i = arr.length - 1; i >= 0; i--) {
    yield arr[i];
  }
}
```

- The return value is an iterator object that may be used in a for loop in the normal way (albeit with abnormal results)

Maps

- A `Map` is a collection of key value pairs

- It differs from a regular object insofar as:
  - its keys can be of any type
  - it is iterable
  - it may perform better for adding/removing pairs

- Maps are particularly useful for random access; i.e. when you need to obtain a given record from many (arrays require an exhaustive/binary search)

Sets

- A `Set` is an unordered (non-indexed) collection of unique values

- NB: it is the reference, and not the content of the referenced object, that is used to determine uniqueness

- Sets may be used to generate unions, intersections, and differences but is limited in so doing to primitives (see above)

Classes

- A `class` is a syntactic layer on JS's prototype-based object orientation, e.g.:

```
class Circle {
  constructor(radius) {
    this.radius = radius;
  }
  circumference() {
    return 2 * Math.PI * this.radius;
  }
}
```

- Inheritance (`extends`), getters and setters (properties), and `static` members are all supported

Modules

- A module is a JS file that exports variables/functions/ classes to be imported into some other module or script, e.g.:

```
// mod1.js
export const greeting = 'Hello';

// script.js
import {greeting} from './mod1.js';
```

- Default exports and aliases are supported too

Summary

- A recap of the ES6 features covered here:
  - template literals: strings with placeholders
  - default parameters
  - block scope: `let` and `const`
  - spread: `...` to unpack
  - rest: `...` again; to pack
  - destructuring: obj props/array elements to variables
  - arrow functions: (params) => body
  - enhanced object props
  - generators: custom iterator
  - maps: key value pair collection
  - sets: collection of unique values
  - classes: like Java; syntactic sugar
  - modules: import and export