ES6

ECMAScript6 (Javascript 2015 specification) syntax and information

Declarations

```
const name = 'yourName';
```

Declares a read-only named constant

```
let index = 0;
```

Declares a block scope local variable, optionally initializing it to a value

Arrays

```
arr.copyWithin(target, start[, end = this.length])
```

Copies the sequence of array elements within the array to the position starting at target

```
arr.entries()
```

Returns a new Array Iterator object that contains the key/value pairs for each index in the array

```
arr.find(callback[, thisArg])
```

Executes the callback function once for each element. If such element is found, it returns the value of that element. Otherwise, returns undefined

```
arr.fill(value[, start = 0[, end = this.length]])
```

Fills all the elements from a start index to an end index with a static value

```
arr.findIndex(callback[, thisArg])
```

Returns an index in the array, if an element satisfies the given testing function. Otherwise -1 is returned

```
arr.keys()
```

Returns a new Array Iterator that contains the keys for each index in the array

```
arr.values()
```

Returns a new Array Iterator object that contains the values for each index in the array

Proxies and Promises

```
var p = new Proxy(target, handler);
```

The Proxy object is used to define custom behavior for fundamental operations (e.g. property lookup, assignment, enumeration, function invocation, etc)

```
new Promise(function(resolve, reject) { });
```

A Promise is used in asynchronous computations to represent an operation that hasn't completed yet, but is expected in the future

```
p.catch(reason => rejection);
```

The catch() method returns a Promise and deals with rejected cases only

```
p.then(value => fulfillment, reason => rejection);
```

The then() method returns a Promise. It takes 2 arguments: callback for the success & failure cases

```
Promise.all([p1, p2, p3]).then(values => console.log(values));
```

The Promise.all(iterable) method returns a promise that resolves when all of the promises in the iterable argument have resolved, or rejects with the reason of the first passed promise that rejects

```
Promise.race([p1, p2, p3]);
```

The race function returns a Promise that is settled the same way as the first passed promise to settle. It resolves or rejects, whichever happens first

Template Strings

```
`Hello, my name is ${firstName}`
```

Using the `${}` syntax, strings can embed expressions

Modules

```
import name from 'module-name';
```

```
import * as name from 'module-name';
```

```
import {foo, bar} from 'module-name';
```

To import functions, objects or primitives exported from an external module. These are the most common types of importing

```
export { myFunction };
```

```
export const name = 'yourName';
```

```
export default myFunctionOrClass
```

To export functions, objects or primitives from a given file or module

Functions and Classes

```
class name [extends] { }
```

The class declaration creates a new class using prototype-based inheritance

```
singleParam => { statements }
```

```
() => { statements }
```

```
(param1, param2) => expression
```

Arrow function expression. Shorter syntax & lexically binds the 'this' value. Arrow functions are anonymous

```
function* name(params) { statements }
```

function* declaration defines a generator function. Generators are functions which can be exited and later re-entered. See yield and yield* for more info

```
function(a, b, ...theArgs) { }
```

The rest parameter syntax allows us to represent an indefinite number of arguments as an array

Left-hand-side Expressions

```
myFunction(...iterableObject);
```

```
[...iterableObject, 4, 5, 6]
```

The spread operator allows an expression to be expanded in places where multiple arguments (for function calls) or multiple elements (for array literals) are expected

```
super([arguments]);
```

```
super.functionOnParent([arguments]);
```

The super keyword is used to call functions on an object's parent. When used in a constructor, super appears alone and must be used before the this keyword can be used

```
new.target
```

The new.target property detects whether a function or constructor was called using the new operator