ES6 Cheat Sheet

const/let instead of var

Using 'var' has serious scoping issues. Do not use 'var' at all anymore! Instead use 'const' and 'let'. 'Let' can be re-assigned, const 'cannot'. In a functional JS world, you will deal with a lot immutable data and very rarely need to re-asssign variables. Therefore you will be using 'const' a lot. Common linter rules will check for this behavior.

```
// ES5
var foo = 1;
foo = 2;

// ES6
let foo = 1;
foo = 2;

const bar = 1;
bar = 2; // Uncaught TypeError: Assignment to constant variable.
```

Template literals

```
// ES5
var name = "John";
console.log('Hello ' + name); // Hello John

// ES6
const name = "John";
console.log(`Hello ${name}`); // Hello John
```

Spread Operator

Quickly create new objects based on old objects' properties.

```
// ES6
const obj1 = { a: '1' , b: '2'};
const obj2 = { c: '3'};

const result = { ...obj1, ...obj2 } // { a: '1' , b: '2', c: '3'}
```

Destructuring

Quickly get some or all properties out of an object or array.

```
// ES5
var obj = { a: '1' , b: '2' };
var a = obj.a;
var b = obj.b;

// ES6
const obj = { a: '1' , b: '2' };
const { a, b } = obj;
```

Rest Operator

When you don't know which properties exist, or if there a lot of them, you can use the spread operator inside of desctructuring to collect all not-explicitly mentioned properties.

```
// ES6
const obj = { a: '1' , b: '2', c: '3'};
const { a, ...rest };

console.log(rest); // { b: '2', c: '3' }
```

Object Shorthand

When the object property and the variable name you want to pass to it, you can omit the value:

```
// ES5
var obj = { x: x, y: y };

// ES6
const obj = { x, y };
```

Arrow functions

'This' in a regular function is bound to the function itself or the underlying class if it is a class method. An arrow function will keep the this-binding of the surrounding context available inside the arrow function.

```
// ES5
function() {
    var that = this;
    function foo(param) {
        that.bar(param, true);
    }

    function bar(param1, param2) { /**/ }
}

// ES6
function() {
    const foo = param => this.bar(param, true);

    const bar = (param1, param2) => { /**/ }
}
```

In arrow functions you can leave out the parentheses around the argument if there is only one argument. Also you can leave out brackets {} if the function contains only one expression to be returned. All three arrow functions are identical:

```
const foo = (someString)  => { return someString.toUpperCase()}
const foo = someString  => { return someString.toUpperCase()}
const foo = someString  => someString.toUpperCase()

// foo('bar') === 'BAR'
```

Map

```
// ES5
var fruits = ['Apples', 'Bananas', 'Oranges'];
for (var i = 0; i < fruits.length; i++) {
    console.log('I like ' + fruits[i]);
};

// ES6
const fruits = ['Apples', 'Bananas', 'Oranges'];
fruits.map(fruit => console.log(`I like ${fruit}`));
```

Filter

Filter creates a new array based on the old array elements. For each element the expression inside the filter callback is called. If the expression results true, it will be kept. If the expression results false, it will be omitted.

```
// ES5
var fruits = [ { name: 'Apples' }, { name: 'Bananas' } ];
var result = [];

for (var i = 0; i < fruits.length; i++) {
    if (fruits.name === 'Apples') {
        result.push(fruits[i]);
    }
}

console.log(result); // [ { name: 'Apples' } ]

// ES6
const result = [ { name: 'Apples' }, { name: 'Bananas' } ]
    .filter(fruit => fruit.name === 'Apples');

console.log(result); // [ { name: 'Apples' } ]
```

import/export

With ES6 there are two kinds of exports. Default exports and named exports.

```
// file1.js
export default foo = 'foo';
export bar = () => { alert('bar') };

// file2.js
import foo, { bar } from './file1.js';
```

Classes

ES6 classes are simply syntactic sugar for objects/functions.

```
// ES5
function MyClass(x, y) {
    this.x = x;
    this.y = y;
}

MyClass.prototype.toString = function () {
    return '(' + this.x + ', ' + this.y + ')';
}

// ES6
class MyClass {
    constructor(x, y) {
        this.x = x;
        this.y = y;
    }

    toString = () => `(${this.x}, ${this.y})`;
}
```