Destructuring Objects as functions parameters in ES6

1. **Allow function to accept an object**

// ES5

function myFunc(opts) {

var name = opts.name;

var age = opts.age;

}

myFunc({ name: 'John', age: 25 });

The previous example can be rewritten like so in ES6

// ES6

function myFunc({name, age}) {

}

1. **Default Values**

This usually involves checking if the value is undefined or using some sort of assign function:

// ES6

function myFunc({name = 'Default user', age = 'N/A'}) {

}

// It can also be split onto multiple lines

function myFunc({

name = 'Default user',

age = 'N/A'

}) {

}

1. **Allowing the configuration object to be optional**

If you're happy with the default values then you would expect to just call the function with no arguments:

myFunc(); // error!

The problem here is that behind the scenes our opts object is undefined so trying to reference opts.name or opts.age will throw an error.

The way to circumvent this is to set a default value for the configuration object itself:

function myFunc({name = 'Default user', age = 'N/A'} = {}) {

}

Above can be seen as extracting values from an empty object which is supposed to contain two properties.

Here's how you might do that in ES5 land:

function myFunc() {

var opts = arguments[0] === undefined ? {} : arguments[0];

var name = opts.name === undefined ? 'Default user' : opts.name;

var age = opts.age === undefined ? 'N/A' : opts.age;

}