**The CommonJS format**

A rundown of module formats (syntax), loaders (like RequireJS), and modules that fit these patterns.

**CommonJS** is an alternative to the AMD format that we looked at in the previous document. CommonJS lets us express dependencies that a loader can load and an API for our modules.

1. ***Requiring packages***: It’s very similar to the AMD pattern – we still call a **require** **function** (supplied by the CommonJS module loader we choose to use) – it’s just assigned to a variable, rather than passed to a function.

Here we require at the top python-style (ish!), rather than wrapping our module in a function and passing its requirements in the function call at the top.

1. ***Exporting our module***: But if we aren’t wrapping our module in an anonymous function, we can’t return an object, can we? No! We need to do it slightly differently.

We add anything we’d like to be accessible by those who load our module to the **‘exports’** object, another thing that (like require) is implemented by our chosen module loader.

**Defining an AMD module**  
*Your loader must implement a* ***'define'*** *function*

We wrap our module code in define(["list of required module files"], function(name of files) {}. 'define' is a function supplied by our AMD module loader library (like RequireJS). It finds the required modules, loads them up, then loads our module. So our entire module is wrapped in an anonymous function. That anonymous function returns an object with the params and functions we want to export (return {cat: cat, functionOne: function1}.

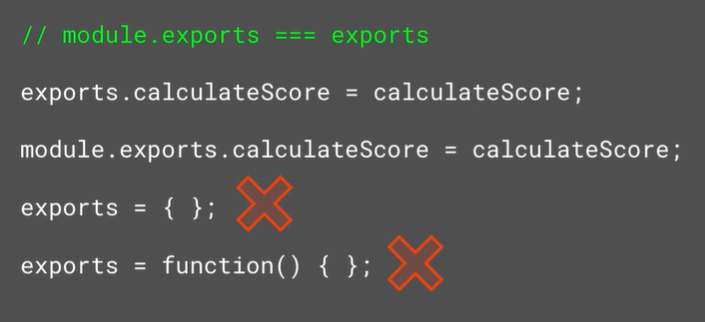
**Defining a CommonJS module**:  
*Your loader must implement* ***'require'*** *and* ***'exports'*** *functions*

Unlike in AMD, we don't wrap our code in an anonymous function. we simply add a line at the top of the script for every library we want to import: 'var player = require(./player.js)'. We can't return an object, because we're not wrapped in a function any more, but we can add things we would like expost to the 'exports' object, e.g. 'exports.printGame = printGame) adds the local 'printGame()' function to the export, and makes it publicly available.

**Export Syntax:**

When we add a variable to the exports object, we’re really adding a variable to the module’s export object, if that makes sense.

We can only pass it variables (which could contain functions or objects or anything) – if we tried to set exports = { }; what we’re doing is replace the exports object with an empty object literal! Identical problem if we assign a function to it directly:



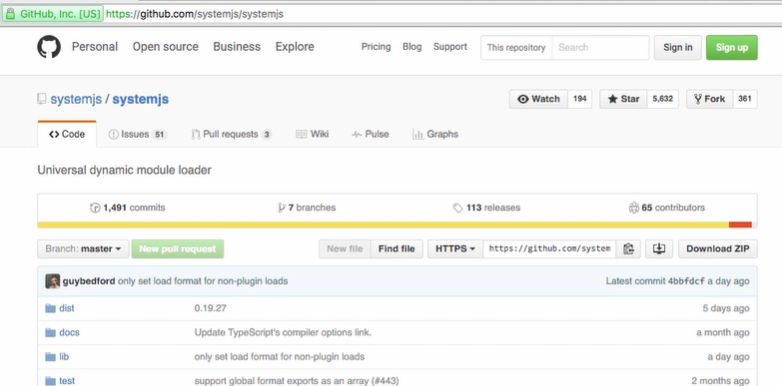
Note that you can actually do the above, you just need to SPECIFICALLY SPECIFY that you’re trying to append to the module’s exports object, not replace exports itself.



**Using a CommonJS module: The SystemJS loader:**

We’ve covered that CommonJS is just a syntax for writing modules. You still need a ‘**module** **loader’** to actually interpret that syntax, and in the case of CommonJS implement ‘require’ and ‘exports’ functionality.

Most of the information for the SystemJS module loader for CommonJS is actually stored on its GitHub page.

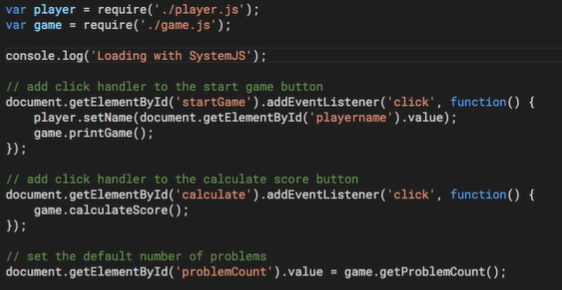


Install this loader just like we installed the AMD loader, remembering that our –save command makes npm save a reference to this library to our package.json:



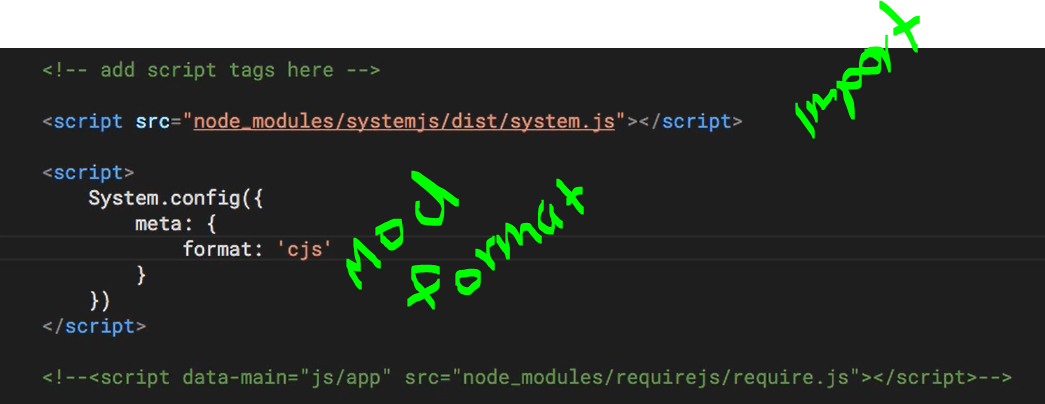
**Implementing System.js in our example:**

Our app.js looks pretty simple now!



Note that we need to fix our index.html, which is currently pointing at the AMD module loader (requireJs).

Note that systemJS requires a bit more inline JavaScript – it’s common practice to configure the library by calling its config function on its System object. There are lots of props you can change, but we’ll simply set the format, so that SystemJS knows it’s loading CommonJS, as this library can load lots of different types and it needs to be specified!.



Note that we finally have to point the module loader to our main module, which loads up everything else. That’s app.js in our case 😊



**Summary**

In module 1 we talked about making basic JS modules. Here we talked about module format vs module loaders, the first being a syntax, the second being a library to interpret that syntax.

We talked about AMD modules with RequireJS.

We also talked about CommonJS modules with SystemJS, most commonly used in server-side code like node.js.

