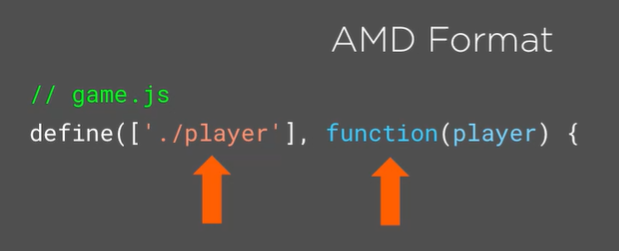
**AMD Format**

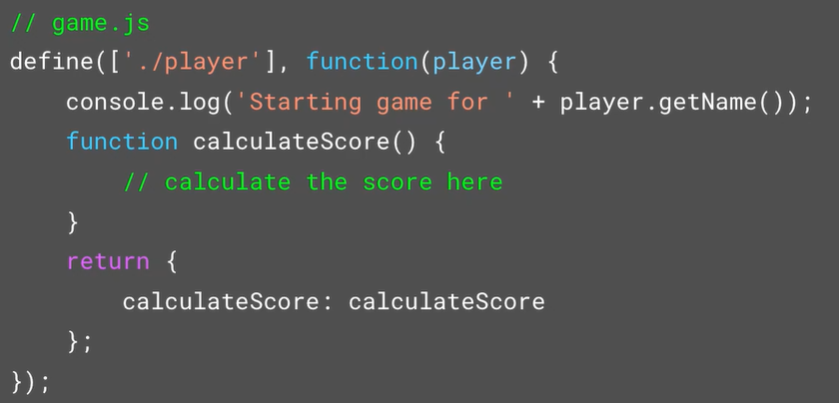
* An AMD module is defined by calling a function called **define().**
* Define is not a built-in JS function, it’ll be implemented by the module loader we choose to include in our project.
* All loaders that support AMD modules will implement a **define** function.

**We pass two arguments:**

* + An array of dependencies that our module relies on.
  + A function that will execute and return a new module.
  + (As an argument to that function, a list of variable names to save our dependencies to).

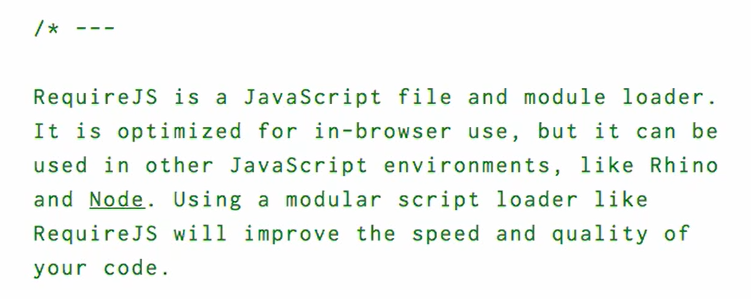


We’re saying we’d like to define a new AMD module. It requires the player module as a dependency, and we want it to be saved to the ‘player’ variable, from where we can access it.



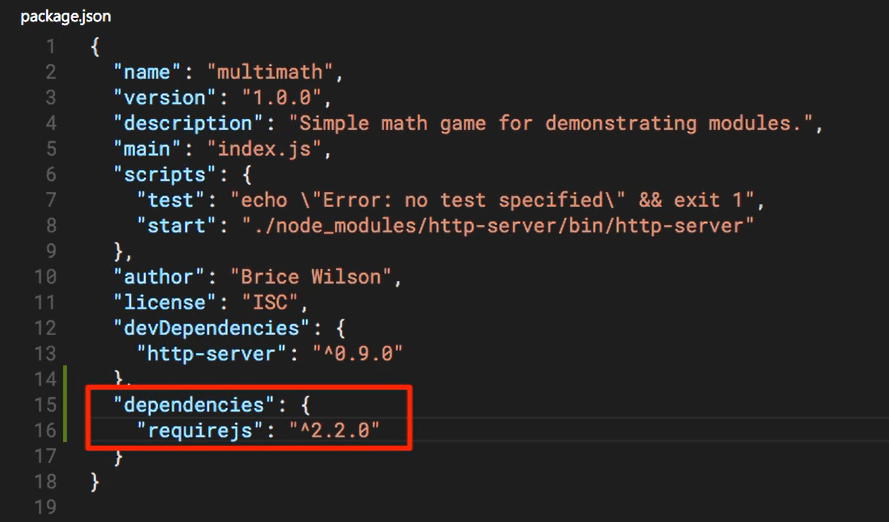
We can import things, hide private vars and packages, and expose what we want to with a returned object. Like the revealing module pattern, but we do not add variables to the global scope, and now we can support modules having their own dependencies without us needing to track that manually.

**RequireJS: A first JavaScript AMD Loader**



RequireJS is one of the module loaders we can use. We can add it to our project with npm, the node.js package manager.

Remember that package.json includes info about our NPM setup & all the packages we require: dev & production. When we type ***‘npm install requirejs –save’*** npm will automatically install our package, save a reference to it in our package.json file.



We now have a module loader! Let’s convert our old module to the AMD format.

**Building an AMD syntax module 😊**

Remember our old player module, which is currently formatted in the revealing module pattern? We’re going to reformat that into an AMD module that RequireJS can interpret and work with.

***Revealing Module Pattern:***

* An anonymous function that’s immediately executed by the trailing (). The variable ‘player’ is available to any HTML page that includes this script with a script tag, and it contains not the function but the object returned by that function, which includes any internal functions you’ve included in your return statement.



***AMD Pattern:***

* The function stays the same, but we change how we wrap it. Remember, we need to wrap the module in the ‘define’ function implemented by our AMD loader, RequireJS now. It takes two arguments: A lost of dependencies, and an anonymous function that would contain a set of params as defined in our list of dependencies.
* We’ve also removed the trailing () so that it’s not called immediately – we don’t want the returned object, we want the function. This way we can call it multiple times to make multiple objects.



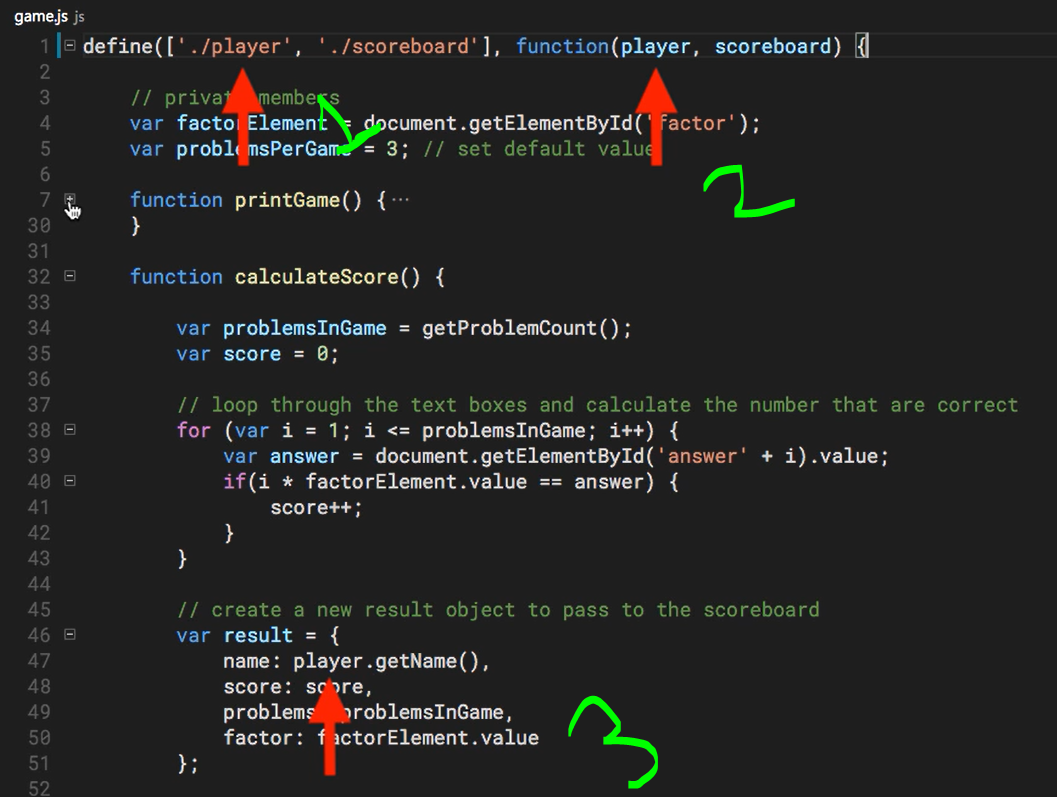
**AMD Pattern:**

* Wrap your code in ‘define’, a function implemented by our AMD package loader.

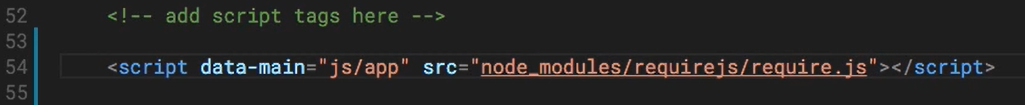


* The bullet pointed list contains a string reference to any packages your module relies on (e.g. ‘./human.js’).
* The anonymous function itself would have an argument for each package you’ve required (e.g. function(humanModule) {).

**Importing a module with our new AMD system:**



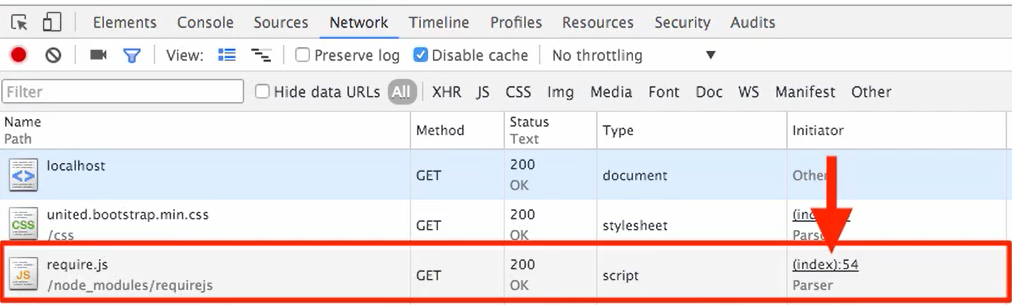
* Here we’re saying ‘I need the player and scoreboard modules defined at the following relative path’. Those are loaded and passed to the anonymous function wrapping our code. This contains all our module code.
* Note that we now define our modules with the define function, which also includes a list of required modules. We no longer need an HTML <script> tag to import things, because it’s all done in the code.
* We do however need to point the HTML page to our main module (app) from which we’ll load and use all the other modules. We also need to pass requireJS (as require.js) to the script tag, to say ‘load it up with this’.



**AMD module loading with RequireJS live, in the real world!**

Now let’s see what happens when the site is accessed!

1. The index.html page is parsed. On line 54 our <script> tag that calls require.js on our local module is read and processed, and require.js is downloaded.



1. Down the list we can see the ‘app’, ‘player’, ‘game’, and ‘scoreboard’ modules loaded in. Note that scoreboard is our main module in the app.js our script tab in (1) points to – but it has app, player, and game as dependencies, so those are loaded first!

Note however that these are loaded in by require.js, line 1958 – we’ve called it and now it’s doing its job, loading in modules 😊



1. In the global namespace (type ‘window’ in the console) you’ll see a require function defined globally. That’s because we call it to pull in our dependencies, so they’re all nested within it, and we don’t need to see a bunch of ‘app’, ‘player’, ‘game’ etc, objects.



1. Woooo! RequireJS is now managing our dependencies and module loading!!!!