Javascript Dynamic Imports

There is a function-like, dynamic form of import that will allow us to play with imports inside modules and classes and will let us work with them together with `async` functions and `awaiting` for async events to complete instead of using `then/catch` blocks for our promise-based async code.

```
<script type="module">
  (async () => {
    try {
      const utilsModuleSpecifier = './utils.js';
      const utilsModule = await import(utilsModuleSpecifier)
      utilsModule.default();
      utilsModule.doStuff();
    }
    catch {
      console.log('Dynamic import failed');
    }
  })();
</script>
```

Modules (either dynamic or static) use defer by default.

ES6 modules run in strict mode, even if you don't write "use strict"; in them. If you're not familiar with strict mode, it can throw unexpected errors and warnings.

You can use import and export in modules.

A Dynamic Module example

The idea behind the example in this section is that the admin module (`admin.js`) will only be loaded if the conditions are met and the user is an administrator.

The idea behind the example in this section is that the admin module (`admin.js`) will only be loaded if the conditions are met and the user is an administrator.

This will save us both on the size of the scripts we load and load time for the

scripts we actually use. The examples are small but you can imagine large admin scripts with lots of functionality.

> The process is as follows:
>
> We define two modules.
>
> One module for users (user.js) for functions and data that relates to users.

```
//user.js
export function isAdmin() {
  console.log('user is an administrator');
  return true
}
```

The second module is for administrator-only functions. We separate them to make each module smaller and more specialized.

```
// admin.js
export function nukeSystem() {
  console.log('system will be nuked in 10 seconds');
  alert('Admin decided to nuke the system');
}
```

Our main script does a few things:

1. Declare that we're using a module
2. Creates an async function
3. Sets up a string to be the user module specifier.
4. Awaits the import of the module with the given specifier
5. Uses a function of the module to check if the user is an administrator
6. If the user is an administrator then we await loading the admin module
7. We run the `nukeSystem` function from the admin module
8. If the user is not an admin we tell the user
9. If the try statement doesn't work, we jump to the catch statement and log an error message to the console

```
<script type="module"> // 1
```

```
  (async () => { //2
    try {
      const userModuleSpecifier = './user.js' './user.js'//3
      const userModule = await import (userModuleSpecifier)// 4
      if (userModule.isAdmin()) { // 5nst adminModuleSpecifier = './admin
        const adminModule = await import (adminModuleSpecifier) // 6
      './admin.js'// 6
        adminModule.nukeSystem() // 7
      } else {
        console.log(`User is not administrator`); `User is not administra
      }
    }
    catch {
      console.log('Something went wrong'); // 9
    }
  })();
</script>
```