

# Intro to ES Modules

Jason Steinshouer

# What are modules?

Modules are an independent self-contained unit of code

# What problem do they solve?

- Make code more maintainable
- Make code more reusable
- Avoid global namespace and global scope issues

# Modules Prior to ECMAScript 6 (ES6)

- Javascript did not have a module system built in
- CommonJS (used in Node.js)
- AMD (requirejs)
- Build tools (Browserify + CommonJS)

# ECMAScript 6 - ES Modules

- New version of Javascript standard release in 2015
- Introduced new standard module system for Javascript
- Node.js build tools (Babel, Webpack)
- Node.js - Currently supports CommonJS (Default) and ES Modules
- Available natively in all major browsers since at least 2018 (Do not need a Node.js build system anymore)

# ES6 Module Basics: Export

- Javascript that can use [export](#) to expose variables and functions to other Javascript modules

```
let myVariable = 0;

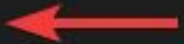
const myfunction = function() {
  console.log("Do something...");
}

export { myFunction, myVariable }; ←
```

# ES 6 Module Basics: Import

- Javascript that can use import to utilise variables and functions from other modules

```
import { myFunction } from "./MyModule.js";  
  
myFunction();
```



# ES 6 Module Basics: Script Tag

- In HTML you can define a `type="module"` attribute for the script tag

```
<script type="module">   
  import $ from "https://esm.sh/jquery";  
  
  $("button.register").on("click", function() {  
    alert("Congrats! You are registered.");  
  });  
</script>
```



# ES 6 Module Basics: Scripts vs Modules

	Scripts	Modules
HTML element	<code>&lt;script&gt;</code>	<code>&lt;script type="module"&gt;</code>
Default mode	non-strict	strict
Top-level variables are	global	local to module
Value of <code>this</code> at top level	<code>window</code>	<code>undefined</code>
Executed	synchronously	asynchronously
Declarative imports ( <code>import</code> statement)	no	yes
Programmatic imports (Promise-based API)	yes	yes
File extension	<code>.js</code>	<code>.js</code>

Source: [https://exploringjs.com/es6/ch\\_modules.html#sec\\_overview-modules](https://exploringjs.com/es6/ch_modules.html#sec_overview-modules)

“Strict mode changes some previously-accepted mistakes into errors.” [Source MDN](#)

# Import / Export In Depth

Import and Export can only be used at the top level

```
// This will fail
✓ if ( thisIsTrue ) {
  |   import { myFunction } from "./MyModule.js";
  |   myFunction();
  |
  }

// This will work
import { myFunction } from "./MyModule.js";

✓ if ( thisIsTrue ) {
  |   myFunction();
  |
  }
```

# Import / Export In Depth: Default export (Only one)

```
1  
/** MyModule.js */  
export default function() {  
  console.log("this is the default");  
}
```

```
/** main.js */  
import myfunction from './MyModule.js';
```

# Import / Export In Depth: Named imports and exports

```
/** MyModule.js */  
export const myVariable = 'I am a string!';  
export const myObject = {  
  doSomething: function() {  
    console.log("this function does things");  
  },  
  myArray: [1,2,3,4]  
}
```

```
/** main.js */  
import { myVariable, myObject } from './MyModule.js';
```

# Import / Export In Depth: Mixed Imports and Exports

```
/** MyModule.js */  
export const myVariable = 'I am a string!';  
export const myObject = {  
  doSomething: function() {  
    console.log("this function does things");  
  },  
  myArray: [1,2,3,4]  
}  
export default function() {  
  console.log("this is the default export");  
}
```

```
/** main.js */  
import myFunction, { myObject, myObject } from './MyModule.js';
```

# Import / Export In Depth: Namespace imports

```
/** MyModule.js */  
export const myVariable = 'I am a string!';  
export const myObject = {  
  doSomething: function() {  
    console.log("this function does things");  
  },  
  myArray: [1,2,3,4]  
}
```

```
/** main.js */  
import * as myModule from './MyModule.js';  
myModule.doSomething();
```

## Import / Export In Depth: Re-exporting

```
/** MyModule2.js */  
export * from './MyModule.js';  
// Renaming: export other_module's foo as myFoo  
export { foo as myFoo, bar } from 'src/other_module';
```

# Import / Export In Depth: Import Maps

[Currently supported in Chrome and Edge](#)

```
<script type="importmap">
  {
    "imports": {
      "BarChart": "./BarChart.js",
      "d3": "https://cdn.skypack.dev/d3@7"
    }
  }
</script>
```

```
import * as d3 from "d3";
```



# Import / Export In Depth: Dynamic Imports

- `import` statement is meant to be static for performance reasons
- `import()` function can be used for loading modules dynamically

```
if ( thisIsTrue ) {  
    import( './MyModule.js' )  
        .then( function( module ) {  
            module.doStuff();  
        } );  
}
```

# You might still want to use a bundler

- Support older browsers
- Performance advantages

Some working examples!

# Resources

- [exploringjs.com](https://exploringjs.com)
- [MDN Module Guide](#)
- [ES 6 Modules In Depth](#)
- <https://web.dev/css-module-scripts/>

Thank you! Questions?