



# JavaScript Modules

## Javascript Plus Session-22



1

# JavaScript Modules





# What is Module?

- ▶ A file - each script file is a module
- ▶ Reusable piece of code that encapsulates implementation details
- ▶ Modules are usually standalone files, don't have to be.





# Benefits of Modules

- ▶ **Compose software:** small packages → complex applications
- ▶ **Isolate components:** provides isolation on entire codebase
- ▶ **Abstract code:** low level implemented code can be stored inside modules, we only call them without knowing the details.
- ▶ **Organized code:** helps for more organized codebase
- ▶ **Reuse code:** the same code can be reused across multiple projects.



DRY: do not repeat yourself



# Brief History

Approach	Runs on	Loaded	Extension
Script	browsers	async	.js
CommonJS	servers	sync	.js .cjs
AMD module	browsers	async	.js
UMD module	browsers and servers	depends	.js
<b>ECMAScript module</b>	<b>browsers and servers(!)</b>	<b>async</b>	<b>.js .mjs</b>



# ES6 Module vs Script

	ES6 Module	Regular script file
namespace pollution	no inside module	global
mode	strict mode	sloppy or loose checking
top-level this	undefined	window
import export	✅ YES (hoisted)	❌ NO
HTML linking	<script type="module">	<script>
download	async	sync
dev env	needs live server	works from local file



# How to

- ▶ **Writing a module**
- ▶ **Using a module**
  - **from js**
  - **from html**





# Writing a Module (ES6 Style)



Declare export

```
mymodule.js > ...  
1  // — mymodule.js —  
2  // named export  
3  export const PI = 3.14;  
4  export const SECONDS_IN_A_DAY = 86400;  
5  export const VERSION = 4.01;  
6  const MINOR_VERSION = 2.26;  
7  
8  export function veryLongNamedFunctionThatDoesSomethingVeryImportant() {  
9    |    return 'veryLongNamedFunctionThatDoesSomethingVeryImportant';  
10 }  
11
```





# Writing a Module (ES6 Style)



Rename export

export as list

```
JS mymodule.js > ...  
12 // rename export  
13 export { SECONDS_IN_A_DAY as SECDAY };  
14 export { veryLongNamedFunctionThatDoesSomethingVeryImportant as doSmt };  
15  
16 // export as list & rename  
17 export { MINOR_VERSION,  
18     VERSION as VER,  
19     veryLongNamedFunctionThatDoesSomethingVeryImportant as doSomething };  
20
```



# Writing a Module (ES6 Style)



default export

```
js mymodule.js > ...  
1  // ——— mymodule.js ———  
2  // default export (!only one)  
3  export default num => {  
4    |   return num * num;  
5  };  
6  // or ! only one default export is allowed  
7  export default 'Module name is mymodule'  
8  |   // don't try to give a name!  
9  export default const moduleName = 'value';
```



# Using a Module (ES6 Style)



declare import

```
myApp.js
1  // — myApp.js —
2  // named import
3  import { SECONDS_IN_A_DAY, doSmt, MINOR_VERSION } from './mymodule.js';
4  console.log(SECONDS_IN_A_DAY);
5
6  // default import
7  import myName from './mymodule.js';
8  console.log(myName);
9
10 // namespace import everything from mymodule with an alias
11 import * as m1 from './mymodule.js';
12 console.log(m1.MINOR_VERSION);
13
```



# Using a Module (ES6 Style)



declare import

```
myApp.js
14 // rename import
15 import { SECONDS_IN_A_DAY as SN_GUN } from './mymodule.js';
16 console.log(SN_GUN);
17
18 // import multi and rename
19 import { veryLongNamedFunctionThatDoesSomethingVeryImportant as f1, VERSION } from './mymodule.js';
20
21 console.log(f1());
22 console.log(VERSION);
```

what about named and default import on a single line?



# Using a Module (ES6 Style)



in html file

```
index.html > html
5  <meta http-equiv="X-UA-Compatible" content="IE=edge" />
6  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
7  <title>Document</title>
8  </head>
9  <body>
10 <script src="myApp.js" type="module"></script>
11 </body>
12 </html>
13
```



1

# JavaScript Code Compatibility



# JS Code Compatibility



How to make our modern code work on older engines that don't understand recent features yet?

Transpilers

Polyfills



# JS Transpilers



## Transpiler

- special piece of software.
- translates source code to another source code.
- can parse modern code.
- rewrite the modern code using older syntax constructs.





# JS Transpilers



- JavaScript before year 2020 didn't have the “nullish coalescing operator”.

```
height = height ?? 100;
```



# JS Transpilers

```
1 // before running the transpiler
2 height = height ?? 100;
3
4 // after running the transpiler
5 height = (height !== undefined && height !== null) ? height : 100;
```



# JS Transpilers

- ASM
- Babel
- CoffeeScript
- Dart
- GrooScript
- JSIL
- Lua JS
- Opal
- PureScript
- Pyjamas
- Scala
- Sweet
- TypeScript
- Traceur
- Whalesong

# JS Pollyfills



## New language features:

- Syntax constructs
- Operators
- Built-in functions

```
Math.trunc(n)
```

# JS Pollyfills



## New language features:

- Syntax constructs
- Operators
- **Built-in functions**

declare the missing  
function



```
Math.trunc(n)
```



# JS Pollyfills

```
1  if (!Math.trunc) { // if no such function
2    // implement it
3    Math.trunc = function(number) {
4      // Math.ceil and Math.floor exist even in ancient JavaScript engines
5      // they are covered later in the tutorial
6      return number < 0 ? Math.ceil(number) : Math.floor(number);
7    };
8  }
```

# JS Polyfills



## libraries of polyfills

- [core js](#): allows to include only needed features.
- [polyfill.io](#) service that provides a script with polyfills.



# JS Code Compatibility



resources showing the support for features

- <https://kangax.github.io/compat-table/es6/>: pure JS.
- <https://caniuse.com/>: browser-related functions.







# Kahoot!



# THANKS!

## Any questions?



# Table of Contents



- ▶ Modules
- ▶ Code Compatibility