Javascript cours n°4

Création et utilisation des modules

Comment organiser son code

- Avant ES6, possibilités assez limitées
  - 1 fichier HTML, *n* fichiers JS
  - L’ordre d’insertion des scripts JS a son importance

Importance de l'ordre des JS

- Exemple :

```
<body>
    <h1>JS course 4</h1>

    <script src="cm4_1.js"></script>
    <script src="cm4_2.js"></script>
</body>
```

```
let hello41 = "I'm a message declared in a cm4_1.js file";

function displayFunction41(message) {
    console.log("From function cm4_1.js I say ", message);
}

displayFunction41(hello41 + " AND " + hello42);
```

```
let hello42 = "I'm a message declared in a cm4_2.js file";

function displayFunction42(message) {
    console.log("From function cm4_2.js I say ", message);
}

displayFunction42(hello42 + " and " + hello41);
```

```
Uncaught ReferenceError: hello42 is not defined
    <anonymous> http://localhost:63342/javatest/td3/cm4_1.js:7
[En savoir plus]
From classic function cm4_2.js I say  I'm a message declared in a
cm4_2.js file and I'm a message declared in a cm4_1.js file
```

Importance de l'ordre des JS

- Peuvent être utilisées peu importe l'ordre de déclaration :
  - Les fonctions
  - Les variables déclarées avec le mot-clé `var`
- … à condition de ne pas être utilisées avant le chargement du fichier JS dans lequel elles sont déclarées
- Dans l'exemple précédent, déclarer `hello42` avec `var` ne change rien...

Conséquences

- Code html allourdi par de nombreuses insertions de fichiers JS
- L’utilisation de librairies externes peut devenir complexe

Solution ES 6 : les modules

- Permet d’importer un fichier / une librairie JS dans un autre JS
- Permet de bien structurer le code
- Dans les fichiers html, on fait uniquement référence aux librairies locales

Solution ES 6 : les modules (ex_1)

- Le fichier « maître » cm4_1 importe ce dont il a besoin depuis la « librairie » cm4_2

- Mot-clé `import`

```
import {hello42, displayFunction42} from "./cm4_2.js";

let hello41 = "I'm a message declared in a cm4_1.js file";

function displayFunction41(message) {
    console.log("From function cm4_1.js I say ", message);
}

displayFunction41(hello41 + " AND " + hello42);
displayFunction42(hello42 + " AND " + hello41);
```

Solution ES 6 : les modules (ex_2)

- La librairie cm4_2 exporte certaines parties du code
- Mot-clé `export`

```
export let hello42 = "I'm a message declared in a cm4_2.js file";

export function displayFunction42(message) {
    console.log("From function cm4_2.js I say ", message);
}
```

Solution ES 6 : les modules (ex_3)

- Le fichier html insère uniqument le fichier maître

```
<body>
    <h1>JS course 4</h1>

    <script src="cm4_1.js" type="module"></script>
</body>
```

- Attention : tout fichier qui importe un module devient lui-même un module

- Pour l'insérer, il faut utiliser l'attribut `type`
- Sinon, erreur :

```
Uncaught SyntaxError: import declarations may only appear at top
level of a module
```

Intérêt des modules

- En entête du fichier cm4_1, on voit tout de suite qu'il nécessite le module cm4_2 pour fonctionner
- Ce n'était pas le cas dans la solution précédente
- Un module va permettre d'écrire des briques de code réutilisables
- Chaque module a son espace de nommage (portée privée au module)

Espace de nommage

- Puisque chaque module a son espace de nommage, il peut arriver qu'il y ait des conflits

- Introduction d'alias lors de l'import

```
import {hello42, displayFunction42 as displayNormal} from "./cm4_2.js";
import {displayFunction42 as displayMajuscules} from "./cm4_3.js";

//...

displayMajuscules(hello41 + " AND " + hello42);
displayNormal(hello42 + " AND " + hello41);
```

Espace de nommage

- Contenu d'un module a une portée privée → fonctions inaccessibles depuis le fichier html

- Sans module

```
<button id="btn1" onclick="fct()">
```

- Avec module :
  - `<button id="btn1">`
  - JS :

```
document.getElementById("btn1")
        .addEventListener("click", fct);
```

Import de la totalité

- Si on utilise la totalité des exports faits dans le module importé, on peut utiliser *
- On spécifie alors un espace de nommage

```
import * as cm42 from "./cm4_2.js";

// ...

displayFunction41(hello41 + " AND " + cm42.hello42);
cm42.displayFunction42(cm42.hello42 + " AND " + hello41);
```

Restrictions

- La plupart des navigateurs interdisent l'import de fichiers js hors d'un serveur
- → il est possible que vous ayez une erreur CORS (Cross-origin Ressource Sharing) si vous exécutez votre fichier html en file://
- Solution : exécuter en localhost

En guise de conclusion

- Les modules sont idéaux pour définir des classes

- Les modules peuvent être regroupés dans des paquetages gérés par `npm`