Javascript 2

Le retour

Les modules

```
// Chaque fichier est un module qui peut exporter ou importer des

// import par défaut d'une fonction foo de la bibliothèque "foo"
import foo
// import nommé d'une fonction bar de la bibliothèque "bar"
import {
// on peut renommer un import nommé
import {
// Et faire les 3 en même temps
import def

// export nommé
export func
    // some code
}

// export par défaut
export defau
    // some code
```

Les exceptions

```
<span class="kd">function</span> <span class="nx">russianRoulette</span><span cla
    <span class="k">if</span><span class="p">(</span><span class="nb">Math</span>
        <span class="c1">// Un problème est survenu...
</span><span class="c1"></span>        <span class="k">throw</span> <span class="
    <span class="p">}</span>
<span class="p">}</span>

<span class="k">try</span> <span class="p">{</span>
    <span class="nx">russianRoulette</span><span class="p">();</span>
    <span class="nx">console</span><span class="p">.</span><span class="nx">log</
<span class="p">}</span>
<span class="k">catch</span><span class="p">(</span><span class="nx">error</span>
    <span class="nx">console</span><span class="p">.</span><span class="nx">error
<span class="p">}</span>
<span class="k">finally</span> <span class="p">{</span>
    <span class="nx">console</span><span class="p">.</span><span class="nx">log</
<span class="p">}</span>
<span class="c1">// Si on catch pas, l'exception remonte la pile
</span><span class="c1">// Si l'exception remonte tout la pile c'est le crash...
```

Javascript est non bloquant (asynchrone)

```
<span class="nx">console</span><span class="p">.</span><span class="nx">log</span>

<span class="c1">// La lecture du fichier est exécuté en tâche de fond par la mac
</span><span class="c1">// L'exécution du code continue
</span><span class="c1"></span><span class="nx">fs</span><span class="p">.</span>
    <span class="c1">// Une fois le fichier lu (quand tout le reste a été traité)
</span><span class="c1"></span>    <span class="nx">console</span><span class="p">
<span class="p">});</span>

<span class="nx">console</span><span class="p">.</span><span class="nx">log</span>
<span class="c1">// => 1 3 2
</span>
```

Le callback hell

```
<span class="c1">// On récupère l'utilisateur courant
</span><span class="c1"></span><span class="nx">fetchUser</span><span class="p">(
    <span class="k">if</span><span class="p">(</span><span class="nx">errUser</spa
        <span class="c1">// On affiche l'erreur s'il y en a une
</span><span class="c1"></span>        <span class="nx">console</span><span class=
        <span class="k">return</span>
    <span class="p">}</span>
    <span class="c1">// On récupère les messages de l'utilisateur courant
</span><span class="c1"></span>    <span class="nx">fetchUserPosts</span><span cla
        <span class="k">if</span><span class="p">(</span><span class="nx">errRight
            <span class="c1">// On affiche l'erreur s'il y en a une
</span><span class="c1"></span>            <span class="nx">console</span><span cl
            <span class="k">return</span>
        <span class="p">}</span>
        <span class="c1">// On affiche les messages
</span><span class="c1"></span>        <span class="nx">console</span><span class=
    <span class="p">})</span>
<span class="p">})</span>
```

La solution: les promesses

```
<span class="c1">// On récupère l'utilisateur courant
</span><span class="c1"></span><span class="k">const</span> <span class="nx">resul
    <span class="c1">// On récupère les messages de l'utilisateur courant
</span><span class="c1"></span>    <span class="p">.</span><span class="nx">then</
    <span class="c1">// On affiche les messages
</span><span class="c1"></span>    <span class="p">.</span><span class="nx">then</
    <span class="c1">// On affiche l'erreur s'il y en a une
</span><span class="c1"></span>    <span class="p">.</span><span class="k">catch</
```

Transformer un callback en promesse

```
function fetchUser() {
    //some code
}

function fetchUserPromise() {
    return new Promise((resolve, reject) => {
        fetchUser((err, data) => {
            if (err) {
                reject(err)
                return
            }
            resolve(data)
        })
    })
}

fetchUserPromise()
    .then(data => {
    .catch(err => {
```

La syntaxe async/await

```
// pour utiliser await il faut que la fonction soit déclarée avec
async function getPostsPromise() {
    try {
       const user = getUserPromise()
       const posts = getPostsPromise()
       return posts
    } catch(e) {
        console.error(e)
        throw e
    }
}

getPostsPromise()
    .then(posts => console.log(posts))
    .catch(e => console.error(e))
// ou avec async/await si on est dans une fonction async
try {
    const posts = getPostsPromise()
    console.log(posts)
```