

# Javascript 2

Le retour

# Les modules

```
// Chaque fichier est un module qui peut exporter ou importer des fonctions, classes, etc.

// import par défaut d'une fonction foo de la bibliothèque "foo"
import foo from "foo"
// import nommé d'une fonction bar de la bibliothèque "bar"
import { bar } from "bar"
// on peut renommer un import nommé
import { foo as foobar } from "foobar"
// Et faire les 3 en même temps
import defaultFoo, { namedBar, secondBar as renamedBar } from "baz"

// export nommé
export function func1() {
    // some code
}

// export par défaut
export default function func2() {
    // some code
}
```

# Les exceptions

```
function russianRoulette() {  
    if(Math.floor(Math.random() * 6) === 0) {  
        // Un problème est survenu...  
        throw new Error("BOOM")  
    }  
}  
  
try {  
    russianRoulette();  
    console.log("t'es vivant")  
}  
catch(error) {  
    console.error("T'es mort")  
}  
finally {  
    console.log("Game over")  
}  
  
// Si on catch pas, l'exception remonte la pile  
// Si l'exception remonte tout la pile c'est le crash...
```

# Javascript est non bloquant (asynchrone)

```
console.log("1")

// La lecture du fichier est exécuté en tâche de fond par la machine virtuelle
// L'exécution du code continue
fs.readFile('/toto.txt', (err, data) => {
  // Une fois le fichier lu (quand tout le reste a été traité), le callback est
  console.log("2")
});

console.log("3")
// => 1 3 2
```

## Le callback hell

```
// On récupère l'utilisateur courant
fetchUser((errUser, user) => {
  if(errUser) {
    // On affiche l'erreur s'il y en a une
    console.error(errUser)
    return
  }
  // On récupère les messages de l'utilisateur courant
  fetchUserPosts(user, (errPosts, posts) => {
    if(errRights) {
      // On affiche l'erreur s'il y en a une
      console.error(errRights)
      return
    }
    // On affiche les messages
    console.log(posts)
  })
})
```

## La solution: les promesses

```
// On récupère l'utilisateur courant
const result = fetchUserPromise()
  // On récupère les messages de l'utilisateur courant
  .then(user => fetchUserPostsPromise(user))
  // On affiche les messages
  .then(posts => console.log(posts))
  // On affiche l'erreur s'il y en a une
  .catch(error => console.error(error))
```

# Transformer un callback en promesse

```
function fetchUser(callback) {  
  //some code  
}  
  
function fetchUserPromise() {  
  return new Promise(function(resolve, reject) {  
    fetchUser((err, user) => {  
      if(err) {  
        reject(err)  
        return  
      }  
      resolve(user)  
    })  
  })  
}  
  
fetchUserPromise()  
  .then(user => console.log(user))  
  .catch(err => console.error(err))
```

# La syntaxe async/await

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

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