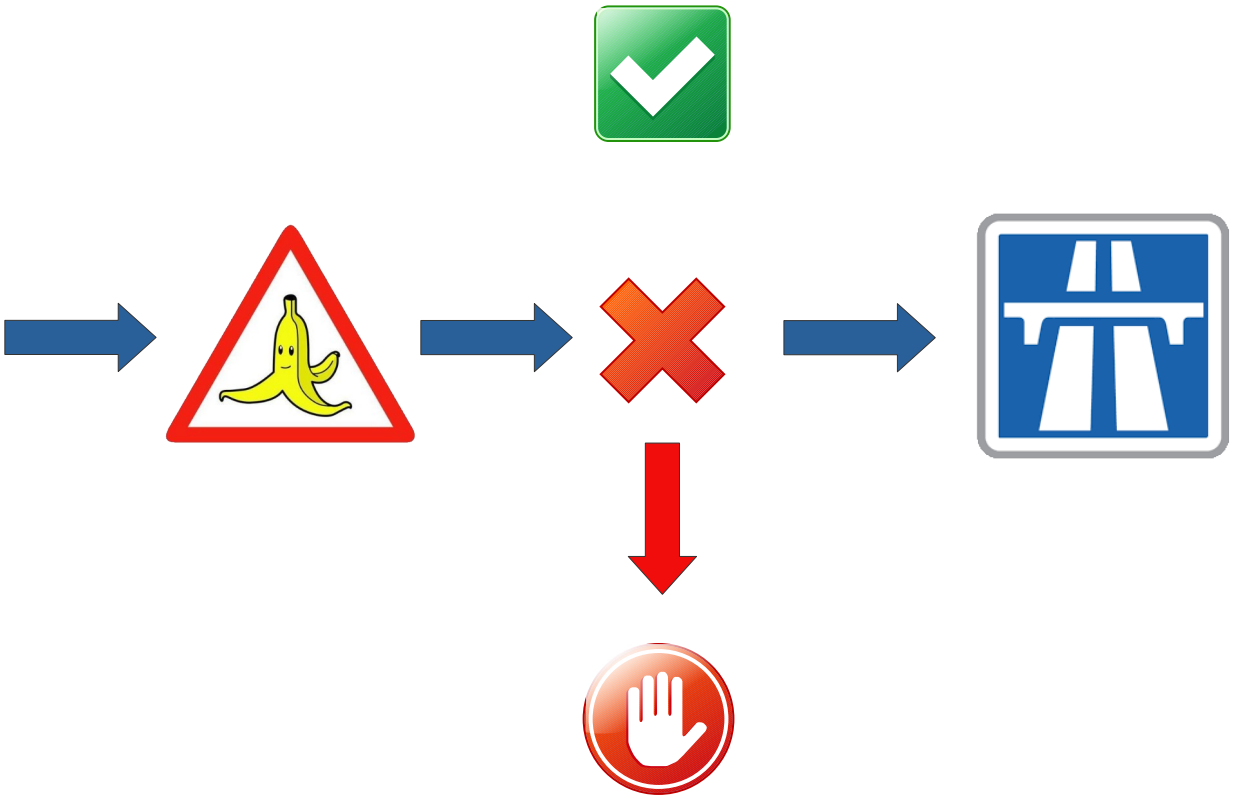
TRY – CATCH - FINAL



date	commentaire
05/01/2020	Création

SOMMAIRE

1 INTRODUCTION

Il n'est pas rare qu'un code puisse planter, c'est-à-dire, s'arrêter de fonctionner.

Il existe de nombreuses raisons pour lesquels un traitement peut ne pas fonctionner, et souvent, l'erreur est récupérable. Ce cours présente le mécanisme qui permet de franchir les passages délicats.

1.1 OBJECTIFS

Ce cours doit permettre de gérer les erreurs dans un programme :

1. Par l'utilisation des exceptions et des mécanismes associés
2. Par la mise en place de ces mécanismes aux bons emplacements
3. Par la compréhension des cas d'utilisation

1.2 PRÉREQUIS

Avoir lu et suivi le cours JavaScript SI4.

1.3 NOTIONS ABORDÉES

Les notions suivantes sont importantes :

- vocabulaire sur les exceptions
- contexte d'utilisation (intérêt, inconvénients)
- mise en œuvre (exemple)

2 PROBLÉMATIQUE

Dans tous les langages, lors d'une interaction ou d'un traitement, le programme ou le système peut détecter une erreur fatale. Dans quel cas trouve-t-on une erreur de ce type ?

- On indique un fichier dans une boite de dialogue (pour l'ouvrir) mais le nom de fichier est faux (ou la clé USB retirée)
- On tente une division par zéro
- On tente une opération arithmétique sur une variable qui n'est pas un nombre (NaN)
- …

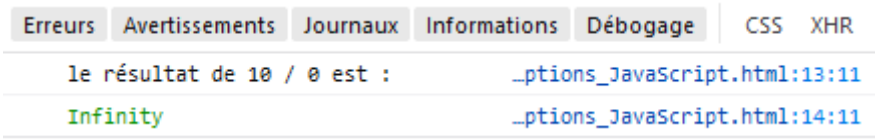
La plupart du temps, le programme s'arrête avec un message d'erreur, ou bien bloque complètement. Dans le cas de pilotes système, on peut même avoir un crash de la machine.

2.1 MAUVAIS EXEMPLE

L'idée d'effectuer une division par zéro semble la plus simple dans tous les langages… mais pas en JavaScript : le code suivant ne plante pas, pire, il donne un résultat juste :

```
let a=0
let b=10
console.log("le résultat de "+b+" / "+b+" est :")
console.log(b/a)
```

Résultat :

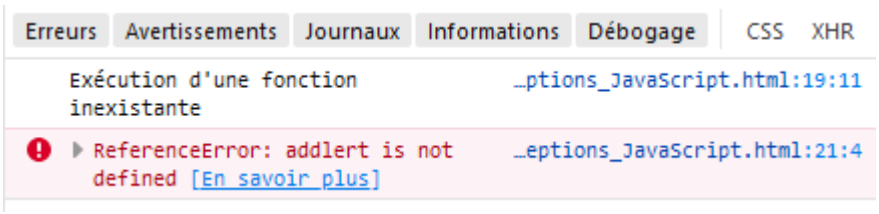


2.2 EXEMPLE CORRECT EN JAVASCRIPT

Voici un code simple qui plantera le programme :

```
console.log("Exécution d'une fonction inexistante")
console.log(addlert("Quoi ?"))
console.log("Programme terminé sans problème")
```

Résultat :

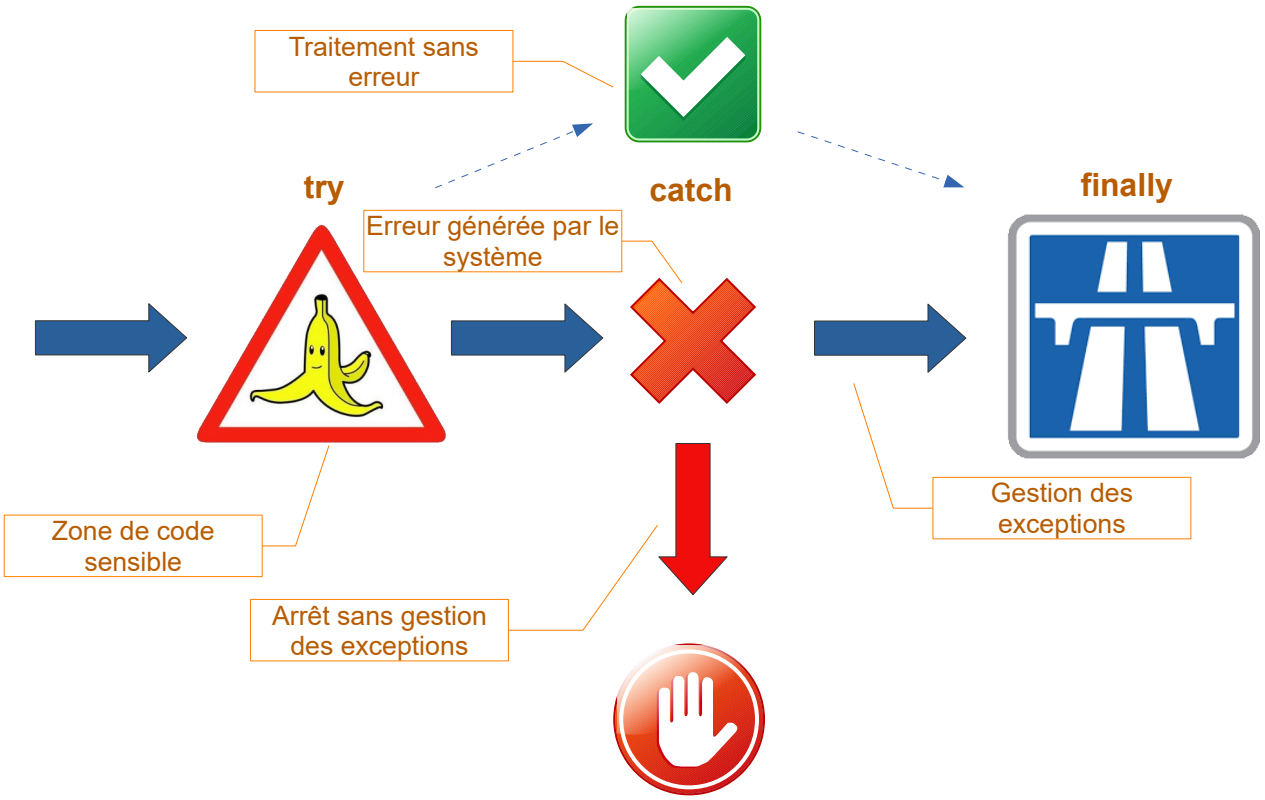


3 GESTION DES EXCEPTIONS

La plupart des programmes gèrent les exceptions : ce sont des événements qui sont déclenchés par le programme (le processeur ou bien l'interpréteur) et qui permettent aux développeurs de reprendre la main sur le traitement en cours.

3.1 SYNOPTIQUE TRY - CATCH

Voici le schéma de fonctionnement des exceptions.



Avant la zone de code sensible, on ouvre un bloc try {…}.

À la sortie de la zone on ferme le bloc et on ouvre un bloc catch {…} qui contient ce qu'il faut faire après une erreur.

On termine avec le reste du code normal.

On peut comparer un try-catch à une condition "si tout va bien, continuer ; sinon gérer le problème puis continuer".

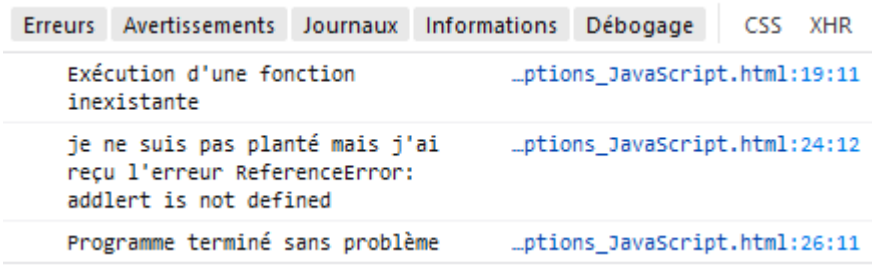
3.2 EXEMPLE DE GESTION

3.2.1 Exemple de base

Voici le même code que précédemment, mais en y ajoutant la gestion des exceptions.

```
// Cet exemple ne plante pas, car si la fonction n'existe pas, l'erreur est gérée
console.log("Exécution d'une fonction inexistante")
try {
	console.log(addlert("Quoi ?"))
}
catch(err) {
	console.log("je ne suis pas planté mais j'ai reçu l'erreur "+err)
}
console.log("Programme terminé sans problème")
```

Résultat :



Comme le montre l'exemple, le bloc à l'intérieur du try ne s'est pas exécuté mais celui à l'intérieur du catch s'est bien déclenché.

On dit que le programme a **levé une exception** (en anglais, on dit "**throw an exception**").

3.2.2 Exemple avancé

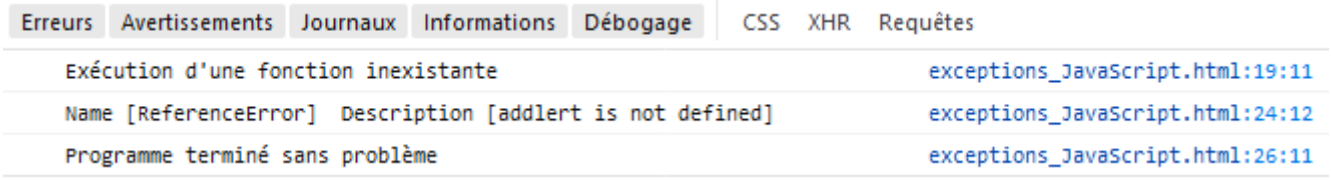
En effet, JavaScript travaille en objet et la classe Error renvoie un objet JavaScript avec 2 propriétés :

- name : contient le nom de l'erreur (la catégorie). De manière standard on trouve :
  - EvalError
  - RangeError
  - ReferenceError
  - SyntaxError
  - TypeError
  - URIError
- message : contient le message associé à la catégorie (une description plus précise de l'erreur)

Voici le même code que précédemment, mais en y ajoutant la gestion de l'objet.

```
// Cet exemple ne plante pas, car si la fonction n'existe pas, l'erreur est gérée
console.log("Exécution d'une fonction inexistante")
try {
	console.log(addlert("Quoi ?"))
}
catch(err) {
	console.log("Name ["+err.name+"]  Description ["+err.message+"]")
}
console.log("Programme terminé sans problème")
```

Résultat :



3.3 LEVÉ D'EXCEPTION

Ce mécanisme de gestion est tellement pratique, qu'il est possible pour le développeur de lever ses propres exceptions.

L'instruction **throw** est utilisée pour cela.

3.3.1 Exemple simple

L'exemple qui suit permet de comprendre comment gérer une saisie et renvoyer une exception "volontairement".

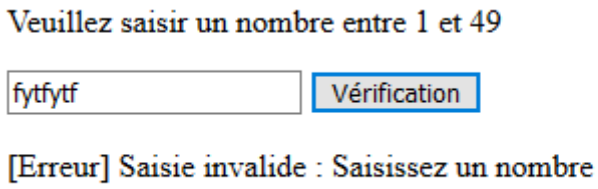
Voici le code :

```
<!DOCTYPE html>
<html lang="fr">
<head>
	<meta charset="UTF-8">
	<title>Exception et throw</title>
	<meta name="viewport" content="width=device-width, initial-scale=1.0">
	<link rel="stylesheet" type="text/css" href="styles.css" />
</head>
<body>
	<p>Veuillez saisir un nombre entre 1 et 49</p>
	<input id="saisieNombre" type="text">
	<button type="button" onclick="VerifierSaisie()">Vérification</button>
	<p id="msgErreur"></p>

	<script>
		// script d'exemple d'usage de throw
		function VerifierSaisie() {
		  let message
		  let x
		  message = document.getElementById("msgErreur")
		  message.innerHTML = ""
		  x = document.getElementById("saisieNombre").value
		  try {
		      if(x == "") throw "Champ vide : saisissez un nombre"
		      if(isNaN(x)) throw "Saisie invalide : Saisissez un nombre"
		      x = Number(x)
		      if(x < 0) throw "Nombre trop Petit"
		      if(x > 49) throw "Nombre trop Grand"
		  }
		  catch(err) {
		      message.innerHTML = "[Erreur] " + err
		  }
		}
	</script>

</body>
</html>
```

Résultat :



Note : on peut aussi écrire **throw new Error("message")** ou même **throw new Error(*JsonObjet*)**.

Pour être compatible avec les autres langages de programmation, je vous recommande l'usage de throw new ... ().

3.3.2 Exemple avancé

Il est donc possible de créer ses propres types d'erreur :

Voici la partie de code modifiée :

```
<script>
      // script d'exemple d'usage de throw
      function VerifierSaisie() {
        let message
        let x
        message = document.getElementById("msgErreur")
        message.innerHTML = ""
        x = document.getElementById("saisieNombre").value
        try {
            if(x == "") throw "Champ vide : saisissez un nombre"
            if(isNaN(x)) throw "Saisie invalide : Saisissez un nombre"
            x = Number(x)
            if(x < 0) throw new RangeError("Nombre trop petit")
            if(x > 49) throw new Error("Nombre trop Grand")
        }
        catch(err) {
            message.innerHTML = "[Erreur] " + err
            console.error("Name ["+err.name+"]  Description ["+err.message+"]")
        }
      }
</script>
```

Choix du type

Notez la possibilité d'utiliser **console.error** pour afficher le message en rouge dans la console

Le type de l'erreur

Le message de l'erreur

Dans cet exemple, si le champ est vide ou ne contient pas un nombre, err.name et err.message seront indéfinis.

3.4 FINALLY

Le dernier point de gestion des erreurs consiste à encadrer le code qui termine officiellement la zone sensible.

Si vous ouvrez un fichier, quel que soit l'erreur rencontrée, il faut ensuite fermer le fichier (pour ne pas garder de flux ouvert inutilement).

Le bloc finally {...} sert à ça.

3.5 AVANTAGES ET INCONVÉNIENTS

3.5.1 Avantages

Les avantages sont importants, car le programme ne peut plus planter et s'arrêter :

- L'utilisateur dispose d'une application fiable
- L'application est plus robuste face aux hackers et personnes malveillantes
- Le développeur peut gérer les erreurs

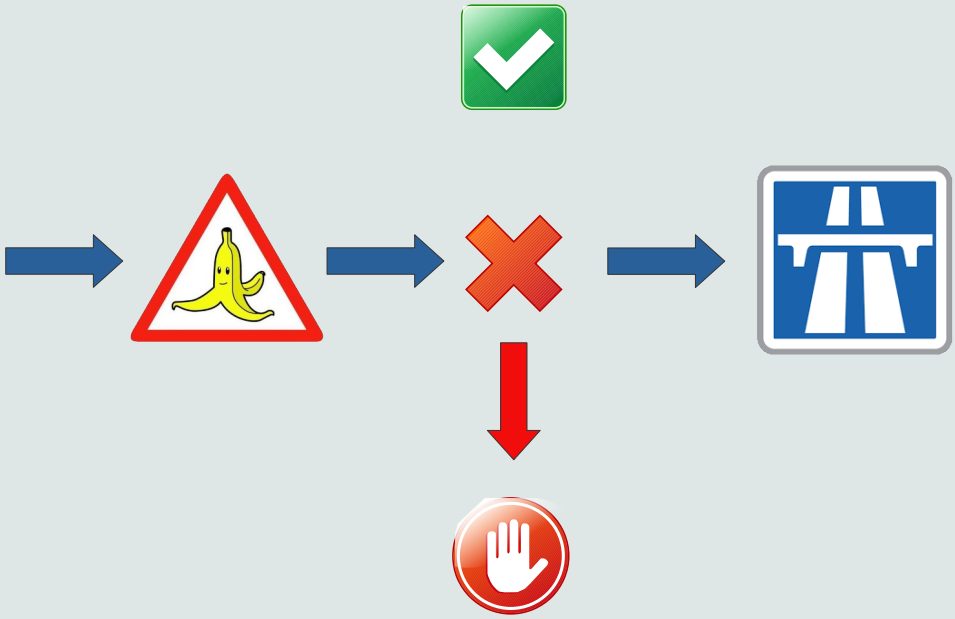
3.5.2 Inconvénients

Il y a quand même un inconvénient majeur : ce système a un impact sur les performances. C'est la raison pour laquelle on ne met pas tout le code dans le bloc try.

4 En résumé

La gestion des exceptions évite que les programmes plantent dans les phases critiques. Pour cela, il existe 4 directives :

- bloc **try {}** : on y place les instructions pouvant renvoyer une erreur
- bloc **catch {}** : on y place les instructions de gestion en cas d'erreur
- bloc **finally {}** : on y place les instructions à exécuter, quoi qu'il arrive (fermeture de flux, suppression de variables, etc.)
- instruction **throw** : active une exception dont le contenu est défini en paramètres. Il est recommandé d'utiliser **throw new *Error*(*message*)** qui ressemble plus aux autres langages de programmation



Lorsque le programme rencontre une erreur, on dit qu'il **lève une exception**.