

# La gestion d'erreur

# Table des matières

<b>I. Contexte</b>	<b>3</b>
<b>II. Les erreurs et exceptions</b>	<b>3</b>
A. Les erreurs et exceptions .....	3
B. Exercice : Quiz .....	5
<b>III. Gérer les erreurs en JavaScript</b>	<b>6</b>
A. Gérer les erreurs en JavaScript .....	6
B. Exercice : Quiz .....	8
<b>IV. Essentiel</b>	<b>9</b>
<b>V. Auto-évaluation</b>	<b>9</b>
A. Exercice .....	9
B. Test .....	12
<b>Solutions des exercices</b>	<b>12</b>

## I. Contexte

**Durée :** 1 heure

**Environnement de travail :** VS Code

**Prérequis :** aucun

### Contexte

La gestion des erreurs est une compétence cruciale pour tout développeur. Les erreurs peuvent survenir à tout moment et leur impact peut être significatif. Une erreur non détectée entraîne rapidement des problèmes importants, tels que des plantages d'applications, des comportements inattendus ou même des failles de sécurité.

Dans ce cours, nous allons explorer les différentes techniques et outils de gestion des erreurs en JavaScript. Nous allons commencer par comprendre les différents types d'erreurs qui peuvent se produire en JavaScript, comment les identifier et les résoudre efficacement. Nous verrons également comment utiliser les blocs try-catch pour gérer les erreurs de manière proactive.

### Attention

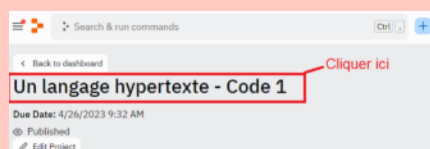
Pour avoir accès au code et à l'IDE intégré de cette leçon, vous devez :

- 1) Vous connecter à votre compte sur <https://replit.com/> (ou créer gratuitement votre compte)
- 2) Rejoindre la Team Code Studi du module via ce lien : <https://replit.com/teams/join/mmurvlgiplxuasordloklllbqskoim-programmer-avec-javascript>

Une fois ces étapes effectuées, nous vous conseillons de rafraîchir votre navigateur si le code ne s'affiche pas.

En cas de problème, redémarrez votre navigateur et vérifiez que vous avez bien accepté les cookies de connexion nécessaires avant de recommencer la procédure.

Pour accéder au code dans votre cours, cliquez sur le nom du lien Replit dans la fenêtre. Par exemple :



## II. Les erreurs et exceptions

### A. Les erreurs et exceptions

#### Fondamental

En JavaScript, les **erreurs** sont un type d'objet créé à l'aide de la syntaxe `new Error()`. Elles sont traitées comme des objets et peuvent être retournées ou placées dans des variables. L'exécuteur JavaScript lui-même renvoie des erreurs.

Une **exception** est une erreur levée. Elle peut être levée par le contexte d'exécution ou par le code avec le mot-clé `throw`, et ne peut plus être manipulée. Elle met ainsi fin aux contextes d'exécution les uns après les autres en remontant, jusqu'à afficher une erreur dans la console. Elle peut être interceptée avec `try`, et l'erreur qu'elles remontent peut être manipulée via `catch` (exception). C'est ce que nous verrons dans ce cours.

En d'autres termes, une erreur est un danger qui pourrait arriver. La classe `Error` nous donnera un outil pour gérer l'erreur si elle se produit. Quand l'erreur se produit, on l'appelle « *exception* ».

## Les erreurs de syntaxe

Les erreurs de syntaxe se produisent lorsqu'une instruction est incorrecte syntaxiquement. Elles sont souvent signalées par des messages d'erreur dans la console du navigateur. **Pour corriger une erreur de syntaxe, il faut modifier votre code.**

### Exemple

Exemple de code avec une erreur de syntaxe :

```
1 const x = 10;
2 if (x === 10 {
3   console.log("x est égal à 10");
4 else {
5   console.log("x n'est pas égal à 10");
6 }
```

[cf.]

L'erreur dans ce code est une parenthèse manquante après le `if`. Le message d'erreur dans la console sera :

```
1 Uncaught SyntaxError: Unexpected token 'else'
```

## Les erreurs d'exécution

Les erreurs d'exécution se produisent lorsqu'une instruction ne peut pas être exécutée correctement. Elles peuvent être causées par des valeurs incorrectes ou inattendues. **Pour corriger une erreur d'exécution, il faut modifier votre code.**

### Exemple

Exemple de code avec une erreur d'exécution :

```
1 let x = 10;
2 let y = x.toUpperCase();
3 console.log(y);
```

[cf.]

Le problème ici est dû au fait que la méthode `toUpperCase` ne peut pas être appelée sur un nombre. Le message d'erreur dans la console sera :

```
1 Uncaught TypeError: x.toUpperCase is not a function
```

## Les erreurs tierces

Les erreurs de bibliothèque se produisent lorsque nous utilisons une bibliothèque tierce qui ne fonctionne pas correctement ou qui renvoie des erreurs. Ces erreurs peuvent être difficiles à diagnostiquer, car elles sont souvent liées à la bibliothèque et non à votre code.

### Exemple

Exemple de code avec une erreur de bibliothèque :

```
1 // Utilisation de la bibliothèque moment.js
2 let date = moment("2022-12-31T23:59:59");
3
4 // Utilisation de la méthode format qui renvoie une erreur
5 let dateStr = date.format("DD-MM-YYYY");
6 console.log(dateStr);
```

[cf.]

Dans cet exemple, nous utilisons la bibliothèque « *moment.js* » pour manipuler des dates. Nous tentons d'utiliser la méthode `format`, mais elle génère une erreur. Dans ce cas, il est important de consulter la documentation de la bibliothèque et de vérifier que nous utilisons correctement ses méthodes.

## Les erreurs de réseau

Les erreurs de réseau se produisent lorsqu'une requête réseau ne peut pas être exécutée correctement. Elles peuvent être causées par une connexion lente, une erreur de serveur ou un serveur inaccessible. Ce sont des erreurs très fréquentes, il est donc fondamental de savoir les gérer.

### Exemple

Exemple de code avec une erreur de réseau :

```
1 // Utilisation de la méthode fetch pour récupérer des données
2 fetch("https://jsonplaceholder.typicode.com/users")
3   .then(response => response.json())
4   .then(data => console.log(data))
5   .catch(error => console.error("Une erreur s'est produite : ", error));
```

[cf. ]

Dans cet exemple, nous utilisons la méthode `fetch` pour récupérer des données à partir d'une API en ligne. Si la requête échoue pour une raison quelconque, le bloc `catch` s'exécutera et affichera un message d'erreur dans la console.

## B. Exercice : Quiz

[solution n°1 p.13]

### Question 1

Qu'est-ce qu'une erreur en JavaScript ?

- ☐ Une fonctionnalité de JavaScript qui permet de contrôler le flux d'exécution du code
- ☐ Un message d'avertissement qui s'affiche lorsqu'une opération est effectuée avec des données non valides
- ☐ Un arrêt de l'exécution du code en raison d'un comportement imprévu ou d'une condition invalide

### Question 2

Les erreurs de syntaxe en JavaScript se produisent lorsque nous avons fait une erreur dans l'écriture de notre code (faute de frappe, mot-clé qui n'existe pas dans le langage, etc.).

- ☐ Vrai
- ☐ Faux

### Question 3

Qu'est-ce qu'une erreur tierce en JavaScript ?

- ☐ Une erreur qui se produit dans notre propre code
- ☐ Une erreur qui se produit dans une bibliothèque tierce que nous utilisons dans notre code
- ☐ Une erreur qui se produit dans le navigateur du client

### Question 4

On peut résoudre une erreur tierce dans notre propre code.

- ☐ Vrai
- ☐ Faux

#### Question 5

Quelle est la différence entre une erreur de syntaxe et une erreur d'exécution en JavaScript ?

- ☐ Une erreur de syntaxe se produit lorsqu'une instruction est mal écrite, tandis qu'une erreur d'exécution se produit lorsqu'une instruction est exécutée avec des données non valides
- ☐ Une erreur de syntaxe se produit lorsqu'une instruction est exécutée avec des données non valides, tandis qu'une erreur d'exécution se produit lorsqu'une instruction est mal écrite
- ☐ Il n'y a pas de différence entre les 2 types d'erreurs

## III. Gérer les erreurs en JavaScript

### A. Gérer les erreurs en JavaScript

#### Try...catch

La structure `try...catch` permet d'exécuter une instruction tout en gérant les erreurs potentielles. Le bloc `try` contient le code à exécuter, tandis que le bloc `catch` contient le code à exécuter en cas d'erreur.

#### Exemple

Exemple de code avec `try...catch`:

```
1 try {
2   let x = 10;
3   let y = x.toUpperCase();
4   console.log(y);
5 } catch (error) {
6   console.log("Une erreur s'est produite : " + error.message);
7 }
```

[cf. ]

On voit ici que le bloc `try` contient 2 instructions. La première affecte la valeur 10 à la variable `x`. La deuxième essaie de convertir la valeur de `x` en majuscules, ce qui provoque une erreur. Le bloc `catch` capture l'erreur et affiche un message personnalisé dans la console.

#### throw

La structure `throw` permet de générer une erreur personnalisée. Elle est souvent utilisée pour signaler des erreurs spécifiques à l'application.

#### Exemple

Exemple de code avec `throw`:

```
1 function diviser(a, b) {
2   if (b === 0) {
3     throw new Error("Division par zéro impossible");
4   }
5   return a / b;
6 }
7
```

```
8 try {
9   let resultat = diviser(10, 0);
10  console.log(resultat);
11 } catch (error) {
12   console.log("Une erreur s'est produite : " + error.message);
13 }
```

[cf. ]

Dans cet exemple, nous avons une fonction `diviser` qui effectue une division entre 2 nombres. Si la valeur de `b` est égale à 0, nous lançons une erreur personnalisée avec `throw`. Lorsque la fonction `diviser` est appelée dans le bloc `try`, elle lève une erreur, car la valeur de `b` est 0. Le bloc `catch` capture l'erreur et affiche un message personnalisé dans la console.

## Les erreurs personnalisées

Il est important de noter que `throw` peut générer n'importe quelle erreur, pas seulement une instance de `Error`. Nous pouvons également créer des classes personnalisées qui étendent la classe `Error` pour définir des erreurs personnalisées avec des propriétés et des méthodes spécifiques à notre application.

### Exemple

Exemple de code avec une classe d'erreur personnalisée :

```
1 class AgeInvalideError extends Error {
2   constructor(message) {
3     super(message);
4     this.name = "AgeInvalideError";
5     this.date = new Date();
6   }
7 }
8
9 function verifierAge(age) {
10  if (age < 18) {
11    throw new AgeInvalideError("L'âge doit être supérieur ou égal à 18 ans");
12  }
13  console.log("L'âge est valide");
14 }
15
16 try {
17   verifierAge(15);
18 } catch (error) {
19   if (error instanceof AgeInvalideError) {
20     console.error(`${error.name} : ${error.message} (${error.date})`);
21   } else {
22     console.error(`Une erreur s'est produite : ${error.message}`);
23   }
24 }
```

[cf. ]

Nous avons créé une classe d'erreur personnalisée `AgeInvalideError` qui étend la classe `Error`. Nous avons ajouté une propriété `name` et une propriété `date` pour ajouter des informations supplémentaires à l'erreur. Dans la fonction `verifierAge`, si l'âge est inférieur à 18, nous lançons une erreur personnalisée en utilisant notre classe `AgeInvalideError`. Dans le bloc `catch`, nous vérifions si l'erreur est une instance de `AgeInvalideError`, et si c'est le cas, nous affichons des informations supplémentaires sur l'erreur.

En utilisant `throw`, nous pouvons personnaliser la gestion des erreurs dans notre application et fournir des informations supplémentaires pour aider à diagnostiquer les problèmes.

## finally

La structure `finally` permet d'exécuter du code quels que soient les résultats des blocs `try` et `catch`. Elle est souvent utilisée pour libérer des ressources.

### Exemple

Exemple de code avec `finally`:

```
1 try {
2   let x = 10;
3   let y = x.toUpperCase();
4   console.log(y);
5 } catch (error) {
6   console.log("Une erreur s'est produite : " + error.message);
7 } finally {
8   console.log("Fin de l'exécution du bloc try...catch.");
9 }
```

[cf. ]

On voit ici que le bloc `finally` contient une instruction qui s'exécutera quoi qu'il arrive, que ce soit après le bloc `try` ou le bloc `catch`.

## B. Exercice : Quiz

[solution n°2 p.14]

### Question 1

La structure `try...catch` permet d'exécuter du code, quels que soient les résultats des blocs `try` et `catch`.

- ☐ Vrai
- ☐ Faux

### Question 2

Quel est le rôle de la structure de contrôle d'erreur `try...catch` en JavaScript ?

- ☐ Elle permet d'exécuter une instruction tout en gérant les erreurs potentielles
- ☐ Elle permet de générer une erreur personnalisée
- ☐ Elle permet d'exécuter du code quels que soient les résultats des blocs `try` et `catch`

### Question 3

Quelle est la structure de contrôle d'erreur en JavaScript qui permet de générer une erreur personnalisée ?

- ☐ `try...catch`
- ☐ `throw`
- ☐ `finally`

### Question 4



Les erreurs personnalisées en JavaScript sont générées avec une seule structure.

- ☐ Vrai
- ☐ Faux

#### Question 5

La structure `catch` est obligatoire dans la structure `try...catch` en JavaScript.

- ☐ Vrai
- ☐ Faux

## IV. Essentiel

Dans ce cours sur la gestion des erreurs en JavaScript, nous avons vu comment traiter les erreurs de syntaxe et d'exécution. Les erreurs de syntaxe sont causées par une erreur dans le code, tandis que les erreurs d'exécution se produisent lorsque le code ne peut pas être exécuté correctement. Pour traiter ces erreurs, nous avons appris à utiliser la structure `try...catch`, qui permet d'exécuter une instruction tout en gérant les erreurs potentielles.

Nous avons également vu comment utiliser la structure `throw` pour générer une erreur personnalisée. Cela nous permet de signaler des erreurs spécifiques à l'application et de fournir des informations supplémentaires pour aider à diagnostiquer les problèmes.

Nous avons également abordé les erreurs tierces. Ces erreurs se produisent dans des bibliothèques tierces que nous utilisons dans notre code. Nous avons vu comment gérer ces erreurs en utilisant les blocs `catch` et `finally`.

Pour finir, nous avons vu comment créer des classes d'erreur personnalisées en étendant la classe `Error`. Cela nous permet de définir des erreurs personnalisées avec des propriétés et des méthodes spécifiques à notre application.

Pour conclure, la gestion des erreurs en JavaScript est une compétence essentielle pour tout développeur. En utilisant les structures de contrôle d'erreur telles que `try...catch` et `throw`, ainsi que les classes d'erreur personnalisées, nous pouvons rendre nos applications plus robustes et fiables pour les utilisateurs.

## V. Auto-évaluation

### A. Exercice

Vous devez réaliser une calculatrice en JavaScript. Vous avez déjà le HTML, le CSS, et une partie de JS déjà prêts, mais vous devez le terminer.

#### Le HTML :

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>Calculatrice en JavaScript</title>
5   <link rel="stylesheet" type="text/css" href="style.css">
6 </head>
7 <body>
8   <h1>Calculatrice en JavaScript</h1>
9   <div class="calculatrice">
10    <label for="num1">Premier nombre:</label>
11    <input type="text" id="num1" name="num1">
12
13    <label for="operator">Opérateur:</label>
14    <input type="text" id="operator" name="operator"/>
15
16    <label for="num2">Deuxième nombre:</label>
17    <input type="text" id="num2" name="num2">
```

```

18
19     <button id="validButton" type="submit">Calculer</button>
20 </div>
21
22 <div id="result"></div>
23
24 <script src="script.js"></script>
25 </body>
26 </html>

```

### Le CSS :

```

1 body {
2   font-family: Arial, sans-serif;
3   background-color: #f5f5f5;
4 }
5
6 h1 {
7   text-align: center;
8   margin-top: 30px;
9 }
10
11 .calculatrice {
12   max-width: 500px;
13   margin: 0 auto;
14   background-color: #fff;
15   padding: 20px;
16   border-radius: 5px;
17   box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
18 }
19
20 label {
21   display: block;
22   margin-bottom: 10px;
23 }
24
25 input[type="text"], select {
26   padding: 10px;
27   border-radius: 5px;
28   border: 1px solid #ccc;
29   font-size: 16px;
30   width: 100%;
31   box-sizing: border-box;
32 }
33
34 button[type="submit"] {
35   background-color: #008CBA;
36   color: #fff;
37   padding: 10px;
38   border: none;
39   border-radius: 5px;
40   font-size: 16px;
41   cursor: pointer;
42   transition: all 0.3s ease-in-out;
43 }
44
45 button[type="submit"]:hover {
46   background-color: #006B87;
47 }

```

```

48
49 #result {
50   text-align: center;
51   margin-top: 20px;
52   font-size: 24px;
53 }

```

### Le JavaScript :

```

1 const btn = document.querySelector('#validButton');
2 const resultDiv = document.querySelector('#result');
3
4 btn.addEventListener('click', (event) => {
5   //Catcher les erreurs possibles pour afficher l'erreur dans resultDiv
6   const num1 = validateNumber(document.querySelector('#num1').value);
7   const num2 = validateNumber(document.querySelector('#num2').value);
8   const operator = validateOperator(document.querySelector('#operator').value, num2);
9   const result = calculate(num1, num2, operator);
10  resultDiv.textContent = result;
11
12 });
13
14 function validateNumber(num) {
15   //Vérifier que num est bien un nombre, sinon lancer une erreur
16 }
17
18 function validateOperator(operator, num2) {
19   //Si l'opérateur est /, vérifier que num2 n'est pas 0, sinon lancer une erreur
20
21   //Vérifier que l'opérateur est bien + - * / sinon lancer une erreur
22 }
23
24 function calculate(num1, num2, operator) {
25   switch (operator) {
26     case '+':
27       return num1 + num2;
28     case '-':
29       return num1 - num2;
30     case '*':
31       return num1 * num2;
32     case '/':
33       return num1 / num2;
34   }
35 }

```

#### Question 1

[solution n°3 p.15]

Implémentez la fonction `validateNumber` qui prend en entrée un nombre et vérifie s'il est valide. Si le nombre est invalide, une erreur est levée avec un message d'erreur personnalisé. Sinon, le nombre est retourné.

#### Question 2

[solution n°4 p.15]

Implémentez la fonction `validateOperator` qui prend en entrée un opérateur et le deuxième nombre et vérifie s'il est valide. Si l'opérateur est invalide ou la division par zéro se produit, une erreur est levée avec un message d'erreur personnalisé. Sinon, l'opérateur est retourné.

Ensuite, *catchez* les erreurs pour les afficher à l'utilisateur.

## B. Test

### Exercice 1 : Quiz

[solution n°5 p.16]

#### Question 1

Quelle est la syntaxe d'une instruction try-catch en JavaScript ?

- ☐ `try { // code à exécuter } finally { // code à exécuter }`
- ☐ `try { // code à exécuter } catch (error) { // code à exécuter }`
- ☐ `try { // code à exécuter } else { // code à exécuter }`

#### Question 2

La structure `try...catch` permet d'exécuter une instruction tout en gérant les erreurs potentielles.

- ☐ Vrai
- ☐ Faux

#### Question 3

La structure `finally` en JavaScript permet d'exécuter du code quels que soient les résultats des blocs `try` et `catch`.

- ☐ Vrai
- ☐ Faux

#### Question 4

Les erreurs personnalisées en JavaScript peuvent être créées en étendant la classe `Error`.

- ☐ Vrai
- ☐ Faux

#### Question 5


La gestion des erreurs est peu utilisée dans le monde professionnel.

- ☐ Vrai
- ☐ Faux

## Solutions des exercices


**Exercice p. 5 Solution n°1****Question 1**

Qu'est-ce qu'une erreur en JavaScript ?

- ☐ Une fonctionnalité de JavaScript qui permet de contrôler le flux d'exécution du code
- ☐ Un message d'avertissement qui s'affiche lorsqu'une opération est effectuée avec des données non valides
- ☒ Un arrêt de l'exécution du code en raison d'un comportement imprévu ou d'une condition invalide
-  Une erreur en JavaScript est un arrêt de l'exécution du code en raison d'un comportement imprévu ou d'une condition invalide. Les erreurs peuvent être causées par des erreurs de syntaxe, des erreurs de logique, des erreurs de type ou des erreurs de données.


**Question 2**

Les erreurs de syntaxe en JavaScript se produisent lorsque nous avons fait une erreur dans l'écriture de notre code (faute de frappe, mot-clé qui n'existe pas dans le langage, etc.).

- ☒ Vrai
- ☐ Faux
-  Vrai. Les erreurs de syntaxe se produisent lorsque le code ne respecte pas les règles de la syntaxe du langage. Cela peut inclure des erreurs telles que des parenthèses manquantes, des guillemets manquants ou des virgules mal placées.


**Question 3**

Qu'est-ce qu'une erreur tierce en JavaScript ?

- ☐ Une erreur qui se produit dans notre propre code
- ☒ Une erreur qui se produit dans une bibliothèque tierce que nous utilisons dans notre code
- ☐ Une erreur qui se produit dans le navigateur du client
-  Une erreur qui se produit dans une bibliothèque tierce que nous utilisons dans notre code. Les erreurs tierces sont des erreurs qui se produisent dans des bibliothèques tierces que nous utilisons dans notre code. Ces erreurs peuvent être difficiles à diagnostiquer, car elles sont souvent liées à la bibliothèque et non à notre code.


**Question 4**

On peut résoudre une erreur tierce dans notre propre code.

- ☐ Vrai
- ☒ Faux
-  Faux. Les erreurs tierces se produisent dans des bibliothèques tierces que nous utilisons dans notre code, donc on ne pourra pas la résoudre nous-mêmes.

**Question 5**

Quelle est la différence entre une erreur de syntaxe et une erreur d'exécution en JavaScript ?

- ☒ Une erreur de syntaxe se produit lorsqu'une instruction est mal écrite, tandis qu'une erreur d'exécution se produit lorsqu'une instruction est exécutée avec des données non valides
- ☐ Une erreur de syntaxe se produit lorsqu'une instruction est exécutée avec des données non valides, tandis qu'une erreur d'exécution se produit lorsqu'une instruction est mal écrite
- ☐ Il n'y a pas de différence entre les 2 types d'erreurs
-  Une erreur de syntaxe se produit lorsqu'une instruction est mal écrite, tandis qu'une erreur d'exécution se produit lorsqu'une instruction est exécutée avec des données non valides. Les erreurs de syntaxe sont détectées avant l'exécution du code, tandis que les erreurs d'exécution sont détectées pendant l'exécution du code.


### Exercice p. 8 Solution n°2

#### Question 1

La structure `try...catch` permet d'exécuter du code, quels que soient les résultats des blocs `try` et `catch`.

☐ Vrai

☒ Faux

 Faux. La structure `finally` permet d'exécuter du code quels que soient les résultats des blocs `try` et `catch`. Le bloc `finally` est souvent utilisé pour libérer des ressources.


#### Question 2

Quel est le rôle de la structure de contrôle d'erreur `try...catch` en JavaScript ?

☒ Elle permet d'exécuter une instruction tout en gérant les erreurs potentielles

☐ Elle permet de générer une erreur personnalisée

☐ Elle permet d'exécuter du code quels que soient les résultats des blocs `try` et `catch`

 Elle permet d'exécuter une instruction tout en gérant les erreurs potentielles. La structure `try...catch` permet d'essayer d'exécuter une instruction tout en capturant les erreurs potentielles. Si une erreur se produit, elle est capturée et gérée dans le bloc `catch`.


#### Question 3

Quelle est la structure de contrôle d'erreur en JavaScript qui permet de générer une erreur personnalisée ?

☐ `try...catch`

☒ `throw`

☐ `finally`

 La structure `throw` permet de générer une erreur personnalisée. Elle est souvent utilisée pour signaler des erreurs spécifiques à l'application.

#### Question 4

Les erreurs personnalisées en JavaScript sont générées avec une seule structure.

☐ Vrai

☒ Faux

- Q Les erreurs personnalisées en JavaScript peuvent être créées en étendant la classe `Error` pour définir des erreurs personnalisées avec des propriétés et des méthodes spécifiques à notre application, mais également avec la structure `throw`.

### Question 5

La structure `catch` est obligatoire dans la structure `try...catch` en JavaScript.

☒ Vrai

☐ Faux

- Q Vous avez toujours besoin d'un bloc `catch`, mais il peut être vide et vous n'avez pas besoin de passer de variable. Si vous ne voulez pas du tout de bloc `catch`, vous pouvez utiliser `try/finally`, mais notez qu'il n'avalera pas les erreurs comme le fait un `catch` vide, donc finalement peu d'intérêt à utiliser ce modèle.

### p. 11 Solution n°3

```
1 function validateNumber(num) {
2     num = parseFloat(num);
3     if (isNaN(num)) {
4         throw new Error('Les deux paramètres d\'entrée doivent être des nombres.');
```

### p. 11 Solution n°4

```
1 const btn = document.querySelector('#validButton');
2 const resultDiv = document.querySelector('#result');
3
4 btn.addEventListener('click', (event) => {
5     try {
6         const num1 = validateNumber(document.querySelector('#num1').value);
7         const num2 = validateNumber(document.querySelector('#num2').value);
8         const operator = validateOperator(document.querySelector('#operator').value, num2);
9         const result = calculate(num1, num2, operator);
10        resultDiv.textContent = result;
11    } catch (error) {
12        resultDiv.textContent = error.message;
13    }
14 });
15
16 function validateNumber(num) {
17     num = parseFloat(num);
18     if (isNaN(num)) {
19         throw new Error('Les deux paramètres d\'entrée doivent être des nombres.');
```

```

30     case '+':
31     case '-':
32     case '*':
33     case '/':
34         return operator;
35     default:
36         throw new Error('Opération invalide.');
```


[cf.]

## Exercice p. 12 Solution n°5

### Question 1


Quelle est la syntaxe d'une instruction try-catch en JavaScript ?

- ☐ try { // code à exécuter } finally { // code à exécuter }
- ☒ try { // code à exécuter } catch (error) { // code à exécuter }
- ☐ try { // code à exécuter } else { // code à exécuter }

 L'instruction try est utilisée pour exécuter un bloc de code susceptible de générer une erreur, tandis que l'instruction catch est utilisée pour gérer les erreurs en spécifiant un bloc de code à exécuter si une erreur se produit.

### Question 2

La structure try...catch permet d'exécuter une instruction tout en gérant les erreurs potentielles.


- ☒ Vrai
- ☐ Faux
-  Vrai. La structure try...catch permet d'essayer d'exécuter une instruction tout en capturant les erreurs potentielles. Si une erreur se produit, elle est capturée et gérée dans le bloc catch.

### Question 3

La structure finally en JavaScript permet d'exécuter du code quels que soient les résultats des blocs try et catch.

- ☒ Vrai
- ☐ Faux




-  Vrai. La structure `finally` permet d'exécuter du code quels que soient les résultats des blocs `try` et `catch`. Elle est souvent utilisée pour libérer des ressources.

#### Question 4

---

Les erreurs personnalisées en JavaScript peuvent être créées en étendant la classe `Error`.

- ☒ Vrai  
☐ Faux


-  Vrai. Les erreurs personnalisées en JavaScript peuvent également être créées en étendant la classe `Error` pour définir des erreurs personnalisées avec des propriétés et des méthodes spécifiques à notre application.

#### Question 5

---

La gestion des erreurs est peu utilisée dans le monde professionnel.

- ☐ Vrai  
☒ Faux

-  Faux. La gestion des erreurs est fondamentale en programmation. Elle évitera des comportements inattendus, et évitera aux utilisateurs de voir des messages techniques. Elle évite aussi que l'application crashe, ce qui est très désagréable pour l'utilisateur.