

Debug

Table des matières

I. Contexte	3
II. Introduction au debug en JavaScript	3
A. Introduction au debug en JavaScript.....	3
B. Exercice : Quiz.....	5
III. Techniques avancées de debug en JavaScript	6
A. Techniques avancées de debug en JavaScript	6
B. Exercice : Quiz.....	9
IV. Essentiel	10
V. Auto-évaluation	10
A. Exercice	10
B. Test.....	12
Solutions des exercices	12

I. Contexte

Durée : 1 h

Environnement de travail : VSCode et Replit

Contexte

Avez-vous déjà été bloqué toute une journée sur votre code parce que vous ne trouvez pas l'erreur qui vous bloque ? C'est une situation courante, mais frustrante... Mais comment passer moins de temps à essayer de comprendre ce qui ne va pas ? Le debug est là pour ça.

Dans ce cours, nous allons explorer les différentes techniques de debug en JavaScript, en commençant par les outils de debug intégrés aux navigateurs web, tels que la console JavaScript et le débogueur. Nous verrons également comment utiliser les points d'arrêt pour suspendre l'exécution de votre code à un point donné.

En pratiquant ces techniques de debug régulièrement, vous pourrez améliorer votre compétence en tant que développeur web, détecter et corriger les erreurs de manière efficace, et offrir une expérience utilisateur optimale.

Attention

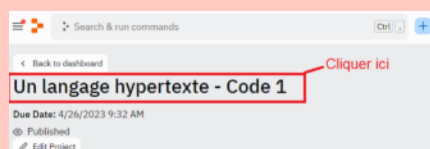
Pour avoir accès au code et à l'IDE intégré de cette leçon, vous devez :

- 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/mmurvlgipxuasordloklbbqskoim-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. Introduction au debug en JavaScript

A. Introduction au debug en JavaScript

Fondamental Lire les messages

Avant même d'apprendre à déboguer, il est fondamental de prendre le temps de lire et d'analyser les messages d'erreurs. Parfois, notre application ne fonctionne pas, on ne comprend pas, mais on ne prend pas le temps de lire ces messages très importants. Or, il est essentiel de prendre le temps de lire et d'analyser ces messages, car ils fournissent des informations précieuses sur votre problématique. Vous pourriez rapidement trouver le problème dans le code, car ils fournissent des indices sur la nature de l'erreur, l'emplacement de celle-ci et, souvent, des suggestions pour la résoudre.

Définition Qu'est-ce que le debug ?

Le debug est le processus de détection et de correction des erreurs dans un programme informatique. En JavaScript, cela signifie détecter et corriger les erreurs dans votre code qui peuvent empêcher votre application web de fonctionner correctement.

Le debug peut être effectué à différentes étapes du développement web, notamment lors de la phase de développement, de test et de déploiement. Les développeurs peuvent utiliser différents outils et techniques de debug pour détecter et corriger les erreurs dans leur code.

Pourquoi est-il important de savoir déboguer son code JavaScript ?

Le debug est une **compétence essentielle** pour tout développeur web, car cela peut vous aider à économiser du temps et de l'argent en évitant des erreurs coûteuses qui pourraient ne pas être détectées avant le lancement d'une application. Le debug est également important pour garantir la sécurité et la fiabilité de votre application.

Si votre code contient des erreurs, cela peut entraîner des problèmes pour les utilisateurs de votre application web. Par exemple, une erreur de syntaxe peut empêcher le chargement de votre application, une erreur de référence peut causer des problèmes de fonctionnement, et une erreur de logique peut affecter les résultats de votre application. En détectant et en corrigeant ces erreurs, vous pouvez garantir que votre application fonctionne correctement et offre une expérience utilisateur optimale.

Méthode Les outils de debug intégrés aux navigateurs web

La plupart des navigateurs web modernes, comme Google Chrome, Firefox et Safari, disposent d'outils de debug intégrés pour JavaScript qui peuvent vous aider à détecter et à corriger les erreurs dans votre code. Voici trois outils :

- La **console JavaScript** est un outil puissant qui vous permet d'afficher des messages, des erreurs et des variables dans la console du navigateur. Vous pouvez utiliser la console pour afficher des messages de debug, des erreurs et des avertissements, ainsi que pour exécuter du code JavaScript directement dans la console.
- Le **débogueur** est un outil qui vous permet de naviguer dans votre code pas à pas, de mettre des points d'arrêt, d'inspecter les variables et les objets, et de surveiller l'exécution de votre code en temps réel. Vous pouvez utiliser le débogueur pour détecter et corriger les erreurs de manière efficace.
- Le **profilage** est un outil qui vous permet d'analyser les performances de votre application web en détectant les goulots d'étranglement et les problèmes de performance. Vous pouvez utiliser le profilage pour optimiser le temps de chargement de votre application et améliorer l'expérience utilisateur.

Complément L'objet console

Tableau des méthodes de l'objet console en JavaScript :

Méthode	Description
console.log()	Affiche des messages dans la console du navigateur.
console.info()	Affiche des informations dans la console du navigateur.
console.error()	Affiche des messages d'erreur dans la console du navigateur.
console.warn()	Affiche des messages d'avertissement dans la console du navigateur.

Méthode	Description
console.clear()	Efface la console du navigateur.
console.table()	Affiche des données dans un tableau dans la console du navigateur.
console.time()	Démarre un minuteur dans la console du navigateur.
console.timeEnd()	Arrête un minuteur dans la console du navigateur.
console.trace()	Affiche une trace de la pile d'appels dans la console du navigateur.
console.assert()	Vérifie une condition et affiche un message d'erreur dans la console du navigateur en cas d'échec
console.group()	Crée un groupe dans la console du navigateur pour afficher des messages connexes.
console.groupEnd()	Ferme le groupe courant dans la console du navigateur.
console.count()	Compte le nombre de fois que cette méthode a été appelée dans la console du navigateur.
console.profile()	Démarre un profilage dans la console du navigateur pour mesurer les performances de votre application.
console.profileEnd()	Arrête le profilage dans la console du navigateur.

B. Exercice : Quiz

[solution n°1 p.13]

Question 1

La console JavaScript est un outil intégré aux navigateurs web pour déboguer votre code JavaScript.

- ☐ Vrai
- ☐ Faux

Question 2

Le profilage est un outil de débogage qui vous permet d'analyser les performances de votre application web.

- ☐ Vrai
- ☐ Faux

Question 3

Le débogueur est un outil qui vous permet de lancer une analyse de performance de votre code.

- ☐ Vrai
- ☐ Faux

Question 4

Le debug permet à l'utilisateur de ne pas rencontrer de bug.

- ☐ Vrai
- ☐ Faux

III. Techniques avancées de debug en JavaScript

A. Techniques avancées de debug en JavaScript

Méthode Utiliser la console JavaScript pour le debug

La console JavaScript est un outil puissant qui vous permet de déboguer votre code en affichant des messages, des erreurs et des variables dans la console du navigateur. Voici comment l'utiliser :

- **Ouvrir la console :** la console peut être ouverte en appuyant sur la touche F12 ou en cliquant avec le bouton droit de la souris et en sélectionnant "Inspecter l'élément" dans le menu contextuel.
- **Afficher des messages dans la console :** vous pouvez afficher des messages de debug en utilisant la méthode `console.log()`. Par exemple, si vous voulez afficher la valeur d'une variable "x", vous pouvez taper `console.log(x)` ; dans la console et la valeur de "x" sera affichée dans la console.
- **Afficher des erreurs dans la console :** si votre code génère une erreur, vous pouvez afficher l'erreur dans la console en utilisant la méthode `console.error()`. Par exemple, si vous voulez afficher une erreur "Erreur de syntaxe" pour une ligne de code spécifique, vous pouvez taper `console.error('Erreur de syntaxe sur la ligne 10')` ; dans la console et l'erreur sera affichée dans la console.
- **Utiliser les autres méthodes de la console :** en plus de `console.log()` et `console.error()`, la console JavaScript dispose d'autres méthodes qui peuvent être utiles pour le debug, telles que `console.info()`, `console.warn()` et `console.table()`. Vous pouvez les utiliser en fonction de vos besoins pour afficher des informations supplémentaires dans la console.

Remarque

Ce cours est sur le debug en JS, il n'y a donc pas besoin de HTML particulier, car on présente uniquement des bouts de code en JavaScript, qui n'interagissent pas avec le HTML.

Exemple Utiliser le console.log() pour corriger un problème

Supposons que vous avez un code JavaScript qui ne fonctionne pas comme prévu et que vous ne savez pas exactement où se trouve le problème. Vous pouvez utiliser `console.log` pour afficher des messages dans la console du navigateur pour comprendre où le code échoue.

Par exemple, supposons que vous ayez une fonction qui doit ajouter deux nombres et retourner le résultat. Le code ressemble à ceci :

```
1 function addNumbers(a, b) {
2   return a + b;
3 }
4
5 var result = addNumbers(2, "3");
6 console.log(result);
```

[cf.]

Dans ce cas, vous vous attendez à ce que la fonction "addNumbers" retourne 5. Cependant, lorsque vous exécutez ce code, vous obtenez un résultat "23" au lieu de 5. Vous ne savez pas où se trouve le problème.

Pour comprendre où le code échoue, vous pouvez ajouter des console.log pour afficher les valeurs des variables à différentes étapes de l'exécution :

```
1 function addNumbers(a, b) {  
2   console.log(a);  
3   console.log(b);  
4   var result = a + b;  
5   console.log("result = " + result);  
6   return result;  
7 }  
8  
9 var result = addNumbers(2, "3");  
10 console.log("resultat final = " + result);
```

[cf.]

Maintenant, lorsque vous exécutez le code, vous verrez les valeurs de "a", "b" et "result" dans la console. Vous pouvez utiliser ces informations pour déterminer que le problème est que la variable "b" est une chaîne de caractères au lieu d'un nombre. Vous pouvez ensuite corriger le code en convertissant "b" en nombre avant de l'ajouter à "a" :

```
1 function addNumbers(a, b) {  
2   console.log(a);  
3   console.log(b);  
4   var result = Number(a) + Number(b);  
5   console.log("result = " + result);  
6   return result;  
7 }  
8  
9 var result = addNumbers(2, "3");  
10 console.log("resultat final = " + result);
```

[cf.]

Maintenant, lorsque vous exécutez le code, vous obtenez un résultat correct de 5.

Nous pouvons le voir dans cet exemple, l'utilisation de console.log est un excellent moyen de comprendre ce qui se passe dans le code JavaScript et de résoudre les problèmes de débogage.

Définition Point d'arrêt

Les points d'arrêt (ou breakpoints en anglais) sont des marqueurs que vous placez à des endroits spécifiques de votre code JavaScript pour suspendre temporairement l'exécution du code. Cela vous permet d'examiner l'état du programme à ce moment-là, d'examiner les valeurs des variables, de parcourir les étapes et de résoudre les problèmes.

En d'autres termes, les points d'arrêt sont un outil de débogage qui vous permet de suspendre l'exécution du code JavaScript à un moment précis et vous aide à comprendre ce qui s'est passé à ce moment-là. Lorsque vous atteignez un point d'arrêt, vous pouvez examiner l'état de l'application à ce stade, y compris les valeurs des variables et des fonctions en cours d'exécution.

Les points d'arrêt sont souvent utilisés pour résoudre des problèmes dans le code JavaScript, en particulier lorsque des erreurs ou un comportement inattendu se produisent dans l'application. En utilisant des points d'arrêt correctement placés, les développeurs peuvent décomposer le code en éléments plus petits et plus faciles à gérer pour mieux comprendre et résoudre les problèmes.

Méthode Utiliser les points d'arrêt pour le debug

Voici comment les utiliser :

- **Placer des points d'arrêt** : vous pouvez placer des points d'arrêt via plusieurs méthodes
 - En cliquant sur la marge de gauche dans l'éditeur de code,
 - En plaçant le mot `"debugger;"` dans votre code.
- **Utiliser le débogueur pour naviguer dans votre code** : une fois que vous avez placé des points d'arrêt, vous pouvez utiliser le débogueur pour naviguer dans votre code pas à pas et inspecter les variables et les objets. Vous pouvez utiliser les boutons de contrôle du débogueur pour exécuter votre code ligne par ligne, sauter des lignes, et reprendre l'exécution de votre code.
- **Examiner les valeurs des variables à différents points dans votre code** : lorsque vous êtes en mode débogage, vous pouvez inspecter les valeurs des variables à différents points dans votre code. Vous pouvez utiliser la fenêtre *"Variables"* du débogueur pour afficher les valeurs actuelles des variables et des objets, ainsi que pour modifier leur valeur à la volée.

En utilisant les points d'arrêt, vous pouvez suspendre l'exécution de votre code à un point donné et inspecter les variables et les objets pour détecter et corriger les erreurs de manière efficace.

Exemple Résoudre le même problème que précédemment, mais avec des points d'arrêts

Pour rappel, voici le code présentant une erreur :

```
1 function addNumbers(a, b) {
2   return a + b;
3 }
4
5 var result = addNumbers(2, "3");
6 console.log(result);
```

Pour comprendre où le code échoue, vous pouvez ajouter un point d'arrêt à l'intérieur de la fonction `addNumbers` pour suspendre l'exécution à ce point. Pour ajouter un point d'arrêt, cliquez simplement sur le numéro de ligne correspondant dans l'éditeur de code ou utilisez le raccourci clavier F9.

```
1 function addNumbers(a, b) {
2   debugger; // Ajout d'un point d'arrêt ici
3   return a + b;
4 }
5
6 var result = addNumbers(2, "3");
7 console.log(result);
```

[cf.]

Maintenant, lorsque vous exécutez le code, l'exécution sera suspendue lorsque le code atteint le point d'arrêt. À partir de là, vous pouvez examiner les valeurs des variables à ce moment-là et déterminer où se trouve le problème.

Dans la console du navigateur, vous pouvez également utiliser des commandes pour examiner l'état de l'application à ce moment-là, telles que la commande *"watch"* pour surveiller la valeur d'une variable ou la commande *"step"* pour exécuter la prochaine instruction. En haut de navigateur, vous avez aussi des flèches qui vous permettent de poursuivre l'exécution du processus.

Complément Utiliser les outils externes de debug

En plus des outils de debug intégrés aux navigateurs web, il existe des outils externes qui peuvent vous aider à déboguer votre code JavaScript de manière plus avancée. Voici quelques exemples :

- **JSFiddle** : une application web qui vous permet de tester et de déboguer votre code JavaScript en ligne, avec la possibilité de partager votre code avec d'autres développeurs.
- **Chrome DevTools** : un ensemble d'outils de développement web intégrés au navigateur Google Chrome, qui comprend un débogueur, une console JavaScript, un inspecteur de code source, un profiler, et bien plus encore.
- **Firebug** : une extension de navigateur pour Firefox qui fournit un ensemble d'outils de développement web, y compris un débogueur, une console JavaScript, un inspecteur de code source, un profiler, et bien plus encore.
- **Visual Studio Code** : un éditeur de code gratuit et open source qui prend en charge le debug pour de nombreuses langues de programmation, y compris JavaScript. Vous pouvez utiliser Visual Studio Code pour déboguer votre code JavaScript en utilisant des points d'arrêt et une console intégrée.

En utilisant des outils externes de debug, vous pouvez améliorer votre processus de développement web en accédant à des fonctionnalités avancées telles que le partage de code en ligne, l'analyse de performances, et une interface de débogage plus riche.

B. Exercice : Quiz

[solution n°2 p.13]

Question 1

La méthode `console.table()` de la console JavaScript vous permet d'afficher des messages de debug dans la console.

- ☐ Vrai
- ☐ Faux

Question 2

Qu'est-ce que `console.log()` ?

- ☐ Une méthode permettant d'afficher des messages dans la console du navigateur
- ☐ Une méthode permettant de suspendre l'exécution du code
- ☐ Une méthode permettant d'ajouter des points d'arrêt

Question 3

Quelle est l'utilité d'un point d'arrêt ?

- ☐ Rendre le code plus rapide
- ☐ Suspendre temporairement l'exécution du code à un endroit spécifique
- ☐ Ajouter une nouvelle fonctionnalité au code

Question 4

Comment peut-on ajouter des points d'arrêt dans une application JavaScript ?

- ☐ En cliquant sur le numéro de ligne dans l'outil de développement du navigateur
- ☐ En tapant « *breakpoint* » dans la console JavaScript
- ☐ En appuyant sur la touche F12

Question 5

Comment utiliser le mot clé `debugger` en js pour placer un point d'arrêt ?

- ☐ Taper la commande `debugger` en console
- ☐ Placer le mot clé `debugger` dans le code JavaScript
- ☐ Placer les mots clés `await debugger` dans le code JavaScript

IV. Essentiel

En conclusion de ce cours sur le debug en JavaScript, nous avons vu que le debug est une compétence essentielle pour tout développeur web, car cela peut vous aider à économiser du temps et de l'argent en évitant des erreurs coûteuses qui pourraient ne pas être détectées avant le lancement d'une application.

Nous avons exploré différentes techniques de debug en JavaScript, en commençant par les outils de debug intégrés aux navigateurs web, tels que la console JavaScript et le débogueur, qui peuvent vous aider à détecter et à corriger les erreurs dans votre code. Nous avons également vu comment utiliser les points d'arrêt pour suspendre l'exécution de votre code à un point donné, et comment utiliser des outils externes tels que JSFiddle et Visual Studio Code pour améliorer votre processus de développement.

En utilisant ces techniques de debug de manière régulière, vous serez en mesure de détecter et de corriger les erreurs dans votre code de manière efficace, garantissant ainsi la fiabilité et la sécurité de votre application web. N'oubliez pas que la pratique régulière du debug est essentielle pour améliorer vos compétences en tant que développeur web.

V. Auto-évaluation

A. Exercice

Vous venez d'être embauché comme développeur dans une petite entreprise, car l'ancien développeur a démissionné. Seulement, le développeur avait commencé le formulaire de contact, mais il ne fonctionne pas, il reste encore des bugs à résoudre, et c'est là que vous intervenez.

Voici le code HTML :

```

1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="UTF-8">
5     <title>Formulaire de contact</title>
6   </head>
7   <body>
8     <h1>Formulaire de contact</h1>
9     <form>
10      <label for="name">Nom :</label>
11      <input type="text" id="name" name="name"><br>
12
13      <label for="email">E-mail :</label>
14      <input type="text" id="email" name="email"><br>
15
16      <label for="message">Message :</label>
17      <textarea id="message" name="message"></textarea><br>
18
19      <button type="submit" id="submit-btn">Envoyer</button>
20    </form>
21
22    <script src="form.js"></script>
23  </body>

```

```
24 </html>
```

Et voici le JavaScript :

```
1 // Ecouter l'événement de clic sur le bouton de soumission
2 document.querySelector("#submit-btn").addEventListener("click", function(event) {
3     event.preventDefault();
4     // Obtenir les valeurs des champs de saisie
5     var name = document.querySelector("#nameInput").value;
6     var email = document.querySelector("#emailInput").value;
7     var message = document.querySelector("#messageInput").value;
8
9     // Vérifier si les champs sont vides
10    if (name === "" && email === "" && message === "") {
11        alert("Veuillez remplir tous les champs");
12        return;
13    }
14
15    // Vérifier si l'adresse e-mail est valide
16    var emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
17    if (!emailRegex.test(email)) {
18        alert("Adresse e-mail invalide");
19        return;
20    }
21
22    // Afficher une alerte pour indiquer que le formulaire a été soumis avec succès
23    alert("Le formulaire a été soumis avec succès !");
24 });
```

Question 1

[solution n°3 p.14]

Voici le mail de votre supérieur :

Objet : Problème à corriger dans le formulaire de contact

Cher développeur,

Je suis ravi de voir que vous travaillez sur notre formulaire de contact pour notre site web. Cependant, nous avons reçu des commentaires de la part des utilisateurs signalant des problèmes avec le formulaire. Voici le problème à corriger :

Je n'ai aucun retour quand je clique sur le bouton, quoi que je fasse, je n'ai aucun message...

Je vous encourage à travailler rapidement sur ces problèmes pour que notre formulaire de contact soit entièrement fonctionnel. N'hésitez pas à me contacter si vous avez des questions ou des préoccupations.

Cordialement,

Question 2

[solution n°4 p.15]

Voici le mail de retour de votre supérieur :

Objet : RE : Problème corrigé dans le formulaire de contact

Cher développeur,

Merci, effectivement, cela fonctionne mieux, en revanche, il me reste toujours un problème :

Les champs de saisie peuvent être soumis même s'ils sont vides. Nous devons nous assurer que les utilisateurs remplissent tous les champs requis avant de soumettre le formulaire.

Tenez-moi au courant quand ce problème sera réglé, pour que nous puissions déployer au plus vite les modifications.

Cordialement,

B. Test

Exercice 1 : Quiz

[solution n°5 p.16]

Question 1

Qu'est-ce que le debug en informatique ?

- ☐ Un processus visant à supprimer les erreurs dans le code
- ☐ Un processus visant à rendre le code plus rapide
- ☐ Un processus visant à ajouter de nouvelles fonctionnalités au code

Question 2

Le profiler est un outil de débogage qui vous permet d'afficher des messages et des erreurs dans la console du navigateur.

- ☐ Vrai
- ☐ Faux

Question 3

Le debug est une compétence inutile pour un développeur web.

- ☐ Vrai
- ☐ Faux

Question 4

Sans outils externes au navigateur, il est impossible de faire du debug.

- ☐ Vrai
- ☐ Faux

Question 5

Comment utilise-t-on un débogueur dans un navigateur web ?

- ☐ En ajoutant des `console.log` dans le code
- ☐ En utilisant des points d'arrêt
- ☐ En tapant "*debugger*;" dans le code


Solutions des exercices

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

La console JavaScript est un outil intégré aux navigateurs web pour déboguer votre code JavaScript.

☒ Vrai

☐ Faux


 Vrai. La console JavaScript est un outil puissant pour déboguer votre code JavaScript en affichant des messages, des erreurs et des variables dans la console du navigateur.

Question 2

Le profilage est un outil de débogage qui vous permet d'analyser les performances de votre application web.

☒ Vrai

☐ Faux


 Vrai. Le profilage est un outil qui vous permet d'analyser les performances de votre application web en détectant les goulots d'étranglement et les problèmes de performance.

Question 3

Le débogueur est un outil qui vous permet de lancer une analyse de performance de votre code.

☐ Vrai

☒ Faux


 Faux. Le débogueur est un outil qui vous permet de naviguer dans votre code pas à pas, de mettre des points d'arrêt, d'inspecter les variables et les objets, et de surveiller l'exécution de votre code en temps réel.

Question 4

Le debug permet à l'utilisateur de ne pas rencontrer de bug.

☐ Vrai

☒ Faux


 Faux. Le debug permet au développeur de résoudre des bugs, l'utilisateur n'a pas connaissance de ce processus de développement.

Exercice p. 9 Solution n°2**Question 1**

La méthode `console.table()` de la console JavaScript vous permet d'afficher des messages de debug dans la console.

☐ Vrai


☒ Faux

 Faux. La méthode `console.table()` de la console JavaScript vous permet d'afficher des données sous forme de tableau dans la console, et non pas des messages de debug.

Question 2

Qu'est-ce que `console.log()` ?


- ☒ Une méthode permettant d'afficher des messages dans la console du navigateur
- ☐ Une méthode permettant de suspendre l'exécution du code
- ☐ Une méthode permettant d'ajouter des points d'arrêt

 `console.log()` est une méthode en JavaScript qui permet aux développeurs d'afficher des messages ou des données dans la console du navigateur. Cette méthode est utile pour déboguer des erreurs, vérifier des valeurs ou suivre le comportement de l'application en affichant des informations dans la console. Les messages et les données affichés dans la console peuvent être de différents types tels que des chaînes de caractères, des nombres, des tableaux, des objets, etc. Par exemple, `console.log(toto)` affichera en console la valeur de la variable `toto`.

Question 3

Quelle est l'utilité d'un point d'arrêt ?


- ☐ Rendre le code plus rapide
- ☒ Suspendre temporairement l'exécution du code à un endroit spécifique
- ☐ Ajouter une nouvelle fonctionnalité au code

 Les points d'arrêt sont utilisés pour comprendre ce qui se passe dans le code à un moment précis et pour examiner l'état de l'application à ce moment-là.

Question 4

Comment peut-on ajouter des points d'arrêt dans une application JavaScript ?


- ☒ En cliquant sur le numéro de ligne dans l'outil de développement du navigateur
- ☐ En tapant « *breakpoint* » dans la console JavaScript
- ☐ En appuyant sur la touche F12

 Pour ajouter un point d'arrêt dans une application JavaScript, vous pouvez cliquer sur le numéro de ligne dans l'outil de développement du navigateur. Cela ajoutera un marqueur rouge qui indique à la machine virtuelle JavaScript de s'arrêter à cette ligne lors de l'exécution du code.

Question 5

Comment utiliser le mot clé `debugger` en js pour placer un point d'arrêt ?

- ☐ Taper la commande `debugger` en console
- ☒ Placer le mot clé `debugger` dans le code JavaScript
- ☐ Placer les mots clés `await debugger` dans le code JavaScript

 Pour utiliser le mot clé `debugger` en JavaScript, il suffit de l'ajouter dans le fichier JavaScript, à l'endroit où l'on souhaite mettre une pause dans l'exécution du code. Lorsque le code atteint cette instruction, il se met en pause et l'inspecteur de débogage du navigateur s'ouvre automatiquement, permettant au développeur d'inspecter l'état des variables et des objets à ce point précis de l'exécution du code.

Les erreurs à corriger :

Correction des getElementById des inputs

- “#emailInput” par “#email”
- “#nameInput” par “#name”
- “#messageInput” par “#message”

Code avec correction :

```

1 // Ecouter l'événement de clic sur le bouton de soumission
2 document.querySelector("#submit-btn").addEventListener("click", function(event) {
3     event.preventDefault();
4     // Obtenir les valeurs des champs de saisie
5     var name = document.querySelector("#name").value;
6     var email = document.querySelector("#email").value;
7     var message = document.querySelector("#message").value;
8
9     // Vérifier si les champs sont vides
10    if (name === "" && email === "" && message === "") {
11        alert("Veuillez remplir tous les champs");
12        return;
13    }
14
15    // Vérifier si l'adresse e-mail est valide
16    var emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
17    if (!emailRegex.test(email)) {
18        alert("Adresse e-mail invalide");
19        return;
20    }
21
22    // Afficher une alerte pour indiquer que le formulaire a été soumis avec succès
23    alert("Le formulaire a été soumis avec succès !");
24 });

```

p. 11 Solution n°4

L'erreur à corriger :

Correction de l'utilisation incorrecte de l'opérateur "&&" en remplaçant par l'opérateur "||" dans la condition de vérification de la saisie des champs

Code avec correction :

```

1 // Ecouter l'événement de clic sur le bouton de soumission
2 document.querySelector("#submit-btn").addEventListener("click", function(event) {
3     event.preventDefault();
4     // Obtenir les valeurs des champs de saisie
5     var name = document.querySelector("#name").value;
6     var email = document.querySelector("#email").value;
7     var message = document.querySelector("#message").value;
8
9     // Vérifier si les champs sont vides
10    if (name === "" || email === "" || message === "") {
11        alert("Veuillez remplir tous les champs");
12        return;
13    }
14
15    // Vérifier si l'adresse e-mail est valide

```

```

16 var emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
17 if (!emailRegex.test(email)) {
18     alert("Adresse e-mail invalide");
19     return;
20 }
21
22 // Afficher une alerte pour indiquer que le formulaire a été soumis avec succès
23 alert("Le formulaire a été soumis avec succès !");
24 }

```

[cf.]

Exercice p. 12 Solution n°5

Question 1

Qu'est-ce que le debug en informatique ?

- ☒ Un processus visant à supprimer les erreurs dans le code
- ☐ Un processus visant à rendre le code plus rapide
- ☐ Un processus visant à ajouter de nouvelles fonctionnalités au code
- ☒ Le debug consiste à identifier et à corriger les erreurs dans le code, telles que les bugs, les exceptions et les plantages.

Question 2

Le profiler est un outil de débogage qui vous permet d'afficher des messages et des erreurs dans la console du navigateur.

- ☐ Vrai
- ☒ Faux
- ☒ Faux. Le profiler est un outil de développement qui permet d'analyser les performances d'une application web pour identifier les zones de code qui peuvent être optimisées.

Question 3

Le debug est une compétence inutile pour un développeur web.

- ☐ Vrai
- ☒ Faux
- ☒ Faux. Le debug est une compétence essentielle pour tout développeur web, car cela peut vous aider à économiser du temps et de l'argent en évitant des erreurs coûteuses qui pourraient ne pas être détectées avant le lancement de l'application.


Question 4

Sans outils externes au navigateur, il est impossible de faire du debug.

- ☐ Vrai
- ☒ Faux
- ☒ Faux, l'outil de debug de chrome ou de votre navigateur permet de faire la majeure partie des opérations de debug.

Question 5

Comment utilise-t-on un débogueur dans un navigateur web ?

- ☐ En ajoutant des `console.log` dans le code
- ☒ En utilisant des points d'arrêt
- ☐ En tapant "*debugger*;" dans le code
-  Les points d'arrêt sont des marqueurs que vous placez dans le code JavaScript pour suspendre temporairement l'exécution du code et examiner l'état de l'application à ce moment-là.