Gestion des bugs : stratégies et outils pour Javascript

Table des matières

I Contexte

Durée : 2h

Environnement de travail : Repl.it

Pré-requis : Aucun

II Notion de bugs

Objectifs

- Connaître la notion de bug ;
- Connaître l'origine de certains bugs.

Mise en situation

En informatique, la situation où tout fonctionne est un cas assez rare : il y a potentiellement beaucoup de raisons pour que peu de choses fonctionnent. Dans ce cas là, on parle de bogue informatique.

Bug (Bogue)

Az Définition

Un bug (bogue en français) est une anomalie de fonctionnement d'un programme informatique.

Origine des bugs

Fondamental

Les bogues peuvent être dus à :

- Des erreurs de syntaxe dans le code source d'un programme.
- Des erreurs de logique dans le code source d'un programme.
- Des erreurs de logique dans le code source des bibliothèques utilisées par le programme.

Erreur de syntaxe

Exemple

```
1 const postcode = 75001
2
3 if (postcode === 75001) {
4   console.log('Premier arrondissement de Paris')
5 }
6 els { // erreur de syntaxe
7   console.log('Autre code postal')
8 }
```

Ici, l'erreur de syntaxe donne lieu à un arrêt du programme.

Erreur de logique

Exemple

```
1 const postcode = 75001
2
3 if (postcode !== 75001) { // Erreur de logique
4   console.log('Premier arrondissement de Paris')
5 }
```

```
else {
  console.log('Autre code postal')
}
```

Ici, l'erreur de logique donne lieu à un résultat incorrect.

Manifestation des bugs dans l'utilisation du programme

Fondamental

Si les erreurs ne sont pas corrigées, plusieurs choses peuvent arriver :

- Le programme peut ne pas s'exécuter du tout.
- Le programme peut commencer à s'exécuter mais ensuite s'arrêter brusquement.
- Le programme peut s'exécuter mais avoir un autre comportement que celui qui est attendu.

Une erreur de logique courante, la boucle infinie

Exemple

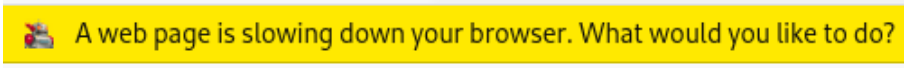
Soit le code suivant supposé afficher tous les entiers naturels impairs inférieurs à 10 :

```
for (let i = 1; i != 10; i = i + 2) {
  console.log(i)
}
```

On se retrouve avec l'exécution de code infinie car la condition `i != 10` n'est jamais atteinte. En effet, on part de 1 et on ajoute 2 à chaque fois : on passera de 9 à 11, sans jamais atteindre 10.

La boucle ne s'arrêtera jamais. Une telle boucle s'appelle **boucle infinie**.

Si ce code est exécuté dans un navigateur, cela peut le ralentir. Dans certains cas, cela peut l'arrêter brusquement.

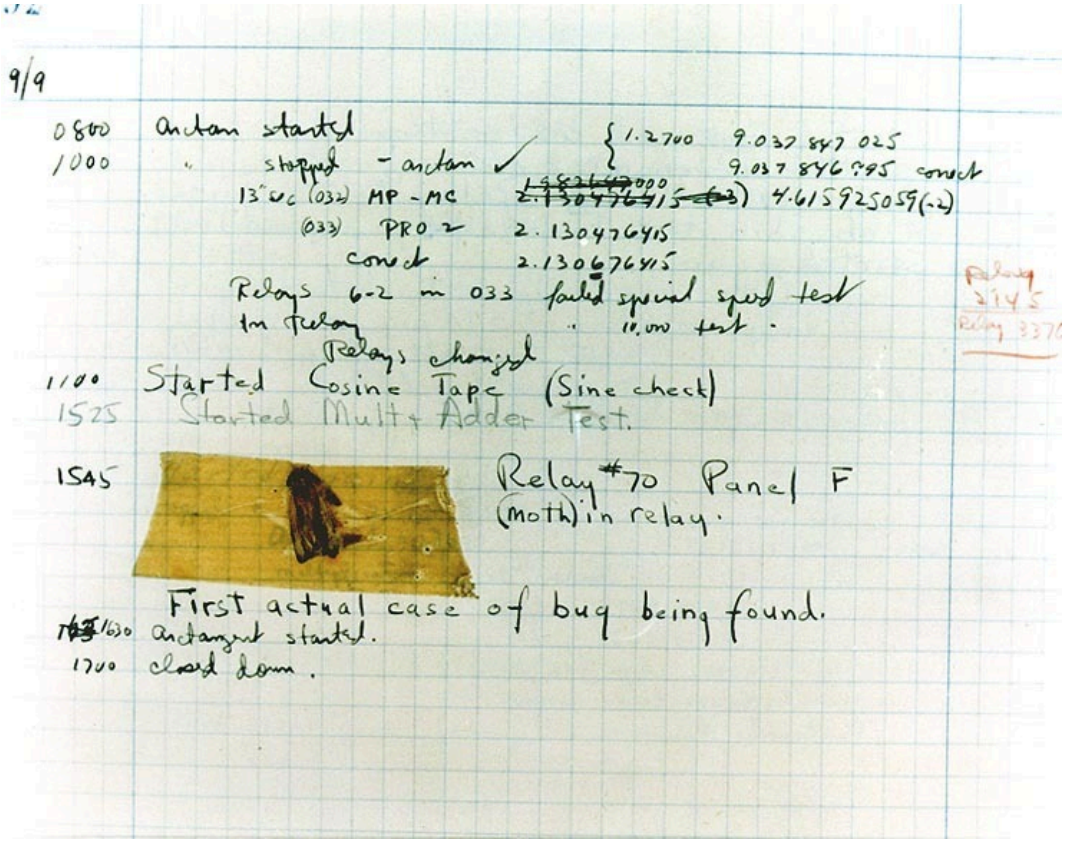


Avertissement de Firefox suite à une boucle infinie

Anecdote : « premier bug » en informatique

Remarque

Lors de la correction d'un bug du Harvard Mark II, ordinateur conservé à la Smithsonian Institution, celui-ci contenait un cafard, en anglais « *bug* ». L'insecte a été conservé et attaché aux notes de Grace Hopper qui travaillait sur cet ordinateur à l'époque.



Premier cas de bogue avéré

Le terme de bug existait cependant avant cette découverte et était déjà utilisé en mécanique.

À retenir

Les bugs sont des anomalies de fonctionnement dues à des erreurs multiples et qui se manifestent par des comportements incorrects divers pour le programme.

III Exercice : Appliquer la notion

Question 1

[solution n°1 p. 35]

Quel est le type de bug présent dans l'exemple suivant (bug dû à une erreur de syntaxe, bug dû à une erreur de logique) ?

Comment corriger le bug ?

```
1 const i = 2
2 if i % 2 {
3   console.log(i + ' est impair')
4 }
```

Question 2

[solution n°2 p. 35]

Quel est le type de bug présent dans l'exemple suivant (bug dû à une erreur de syntaxe, bug dû à une erreur de logique) ?

Comment corriger le bug ?

```
1 const i = 1
2 if (i % 2 !== 0) {
3   console.log(i + ' est pair')
4 }
```

IV Méthodes pour la correction de bugs

Objectif

- Connaître des méthodes pour la correction de bugs.

Mise en situation

Les bugs sont inévitables. Il arrive systématiquement un moment où on doit corriger ces bugs. Suivre une méthode rigoureuse de correction de bugs permet de gagner du temps et de s'assurer que la correction est efficace.

Isoler et minimiser la partie du code contenant le problème

Méthode

Pour mieux comprendre le problème, il est préférable de l'isoler en un exemple simple. Une fois cet exemple isolé, on peut le simplifier le plus possible pour trouver le problème correspondant, s'assurer de l'élément causant le dysfonctionnement et le corriger.

Exemple

On dispose du code suivant avec deux fonctions :

```
1  function fahrenheitToCelsiusFormatting (temperatureF) {
2    // Conversion d'un valeur de degré Fahrenheit à degré Celsius
3    const temperatureC = (temperatureF - 32) * 5 / 9
4  }
5
6  function temperatureConversion (startTempF, nTemperatures, scale) {
7
8    console.log('Début de la conversion des températures ...')
9
10   let temperature = startTempF
11
12   let countTemperature = 0
13
14   while (countTemperature !== nTemperatures) {
15     fahrenheitToCelsiusFormatting(temperature)
16     nTemperatures = nTemperatures + 1
17     temperature = temperature + scale
18   }
19 }
20
21 const startTempF = -459
22 const nTemperatures = 10
23 const scale = 40
24 temperatureConversion(startTempF, nTemperatures, scale)
25
```

L'exécution de ce code ne se termine jamais.

On sent que le problème se situe au niveau de la boucle `while`, qui ne s'arrête jamais, et produit une **boucle infinie**.

```
while (countTemperture !== nTemperatures) {
  fahrenheitToCelsiusFormatting(temperature)
  nTemperatures = nTemperatures + 1
  temperature = temperature + scale
}
```

On peut isoler ces lignes avec le nécessaire pour reproduire le problème ainsi :

```
let startTempF = -459
let nTemperatures = 10
let scale = 40

function fahrenheitToCelsiusFormatting (temperatureF) {
  // Conversion d'un valeur de degré Fahrenheit à degré Celsius
  const temperatureC = (temperatureF - 32) * 5 / 9
}

let temperature = startTempF
let countTemperature = 0

while (countTemperature != nTemperatures) {
  fahrenheitToCelsiusFormatting(temperature)
  nTemperatures = nTemperatures + 1
  temperature = temperature + scale
}
```

On obtient une donc un code bien plus court, produisant le même problème et qui peut être inspecté plus facilement.

Remarque

Un tel exemple est appelé **MWE**, pour *Minimal Working Example*. Cette pratique permet d'isoler précisément le bug, et de s'assurer qu'il n'est pas produit par une instruction située à un autre endroit et n'est pas dépendante du contexte. Pour signaler un bug à d'autres développeurs, il est courant de construire un MWE, qui permet à chacun de reproduire le problème.

Effectuer une recherche efficacement

Méthode

On obtient généralement un **message d'erreur** en présence de bug de syntaxe. Une bonne pratique est d'approfondir la signification de l'erreur et de chercher une solution sur le Web, en se basant sur ce message. Afin de trouver de manière plus rapide ou exacte un résultat, on peut grâce à une certaine syntaxe sur certains navigateurs réaliser des requêtes plus fines.

Dans le cas de *Duckduckgo* par exemple, on peut saisir :

- `'chien chat'`, pour trouver des résultats avec cette chaîne de caractères exacte.
- `chien -chat`, pour trouver des résultats qui font mention de « *chien* » mais qui ne font pas mention de « *chat* ».
- `chien filetype:pdf`, pour trouver des documents pdf qui font mention de « *chien* ».
- `chien site:exemple.fr`, pour trouver des résultats qui font mention de « *chien* » sur exemple.fr.

Plus de détails ici : https://help.duckduckgo.com/duckduckgo-help-pages/results/syntax/

Exemple

Si on tombe en programmant avec JavaScript sur l'erreur suivante :

```
Uncaught TypeError: undefined is not a function
```

Une bonne requête ciblée, pour trouver des éléments de réponse, sera :

```
javascript site:stackoverflow.com 'Uncaught TypeError: undefined is not a function'
```

Qui nous permet d'accéder à des réponses sur StackOverflow, un site d'échanges sur la programmation en ligne.

Par exemple celle-ci qui indique qu'une fonction a sûrement mal été définie : https://stackoverflow.com/questions/13502733/uncaught-typeerror-undefined-is-not-a-function-beginner-backbone-js-applica

Afficher les différentes valeurs des variables

Méthode

Afficher les différentes valeurs des variables de l'exemple isolé permet de mieux les comprendre. Sous JavaScript, on peut réaliser cela avec `console.log`.

Exemple

Soit l'exemple d'une boucle infinie ci-après. On peut arriver à comprendre le problème en affichant la valeur de la variable `countTemperature`.

```
let startTempF = -459
let nTemperatures = 10
let scale = 40

function fahrenheitToCelsiusFormatting (temperatureF) {
  // Conversion d'un valeur de degré Fahrenheit à degré Celsius
  const temperatureC = (temperatureF - 32) * 5 / 9
  console.log(temperatureF, ' °F est équivalent à ', temperatureC, ' °C')
}

let temperature = startTempF
let countTemperature = 0

while (countTemperature !== nTemperatures){
  console.log('countTemperature = ', countTemperature)
  console.log('nTemperatures = ', nTemperatures)
  fahrenheitToCelsiusFormatting(temperature)
  nTemperatures = nTemperatures + 1
  temperature = temperature + scale
}
```

Ici, on obtiendrait en sortie :

```
countTemperature =  0
nTemperatures =  10
-459  °F est équivalent à  -272.77777777777777  °C
countTemperature =  0
nTemperatures =  11
-419  °F est équivalent à  -250.55555555555554  °C
countTemperature =  0
nTemperatures =  12
```

```
-379  °F est équivalent à  -228.33333333333334  °C
countTemperature =  0
nTemperatures =  13
-339  °F est équivalent à  -206.11111111111111  °C
countTemperature =  0
nTemperatures =  14
-299  °F est équivalent à  -183.88888888888889  °C
countTemperature =  0
nTemperatures =  15
-259  °F est équivalent à  -161.66666666666666  °C
countTemperature =  0
nTemperatures =  16
-219  °F est équivalent à  -139.44444444444446  °C
countTemperature =  0
nTemperatures =  17
-179  °F est équivalent à  -117.22222222222223  °C
countTemperature =  0
nTemperatures =  18
-139  °F est équivalent à  -95  °C
countTemperature =  0
nTemperatures =  19
-99  °F est équivalent à  -72.77777777777777  °C
```

Ce qui nous montre que le comportement n'est pas le bon : `nTemperatures` est incrémenté à la place de `countTemperature`.

Commenter temporairement les parties non essentielles

Méthode

Afin de plus rapidement comprendre un bug, on peut commenter les parties de code non essentielles à la résolution du problème.

Cela a l'avantage de :

- réduire le temps d'exécution pour avoir des itérations plus rapide lors du débogage,
- avoir moins de sorties pour mieux se concentrer sur les affichages de débogage.

Exemple

Si on reprend l'exemple précédent, on peut commenter `fahrenheitToCelsiusFormatting` ici pour plus rapidement faire exécuter la boucle et se concentrer sur la sortie de l'affichage du débogage.

```
let startTempF = -459
let nTemperatures = 10
let scale = 40

function fahrenheitToCelsiusFormatting (temperatureF) {
  // Conversion d'un valeur de degré Fahrenheit à degré Celsius
  const temperatureC = (temperatureF - 32) * 5 / 9
  console.log(temperatureF, ' °F est équivalent à ', temperatureC, ' °C')
}

let temperature = startTempF
let countTemperature = 0

while(countTemperature !== nTemperatures){
  console.log('countTemperature = ', countTemperature)
  console.log('nTemperatures = ', nTemperatures)
```

```
  // fahrenheitToCelsiusFormatting(temperature)
  nTemperatures = nTemperatures + 1
  temperature = temperature + scale
}
```

Ici, on obtiendrait uniquement les valeurs des variables à déboguer.

```
countTemperature =  0
nTemperatures =  10
countTemperature =  0
nTemperatures =  11
countTemperature =  0
nTemperatures =  12
countTemperature =  0
nTemperatures =  13
countTemperature =  0
nTemperatures =  14
countTemperature =  0
nTemperatures =  15
countTemperature =  0
nTemperatures =  16
countTemperature =  0
nTemperatures =  17
countTemperature =  0
nTemperatures =  18
countTemperature =  0
nTemperatures =  19
countTemperature =  0
nTemperatures =  20
countTemperature =  0
nTemperatures =  21
countTemperature =  0
nTemperatures =  22
countTemperature =  0
nTemperatures =  23
countTemperature =  0
nTemperatures =  24
...
```

L'affichage est moins pollué, ce qui permet de comprendre plus facilement l'origine du problème.

S'expliquer le problème.

Méthode

Afin de mieux comprendre l'exemple défectueux, on peut essayer d'expliquer le problème. Cela facilite la compréhension de celui-ci et donc sa résolution. Écrire le problème que l'on rencontre peut aussi permettre de mieux le résoudre. Une telle technique peut s'opérer en en 5 étapes :

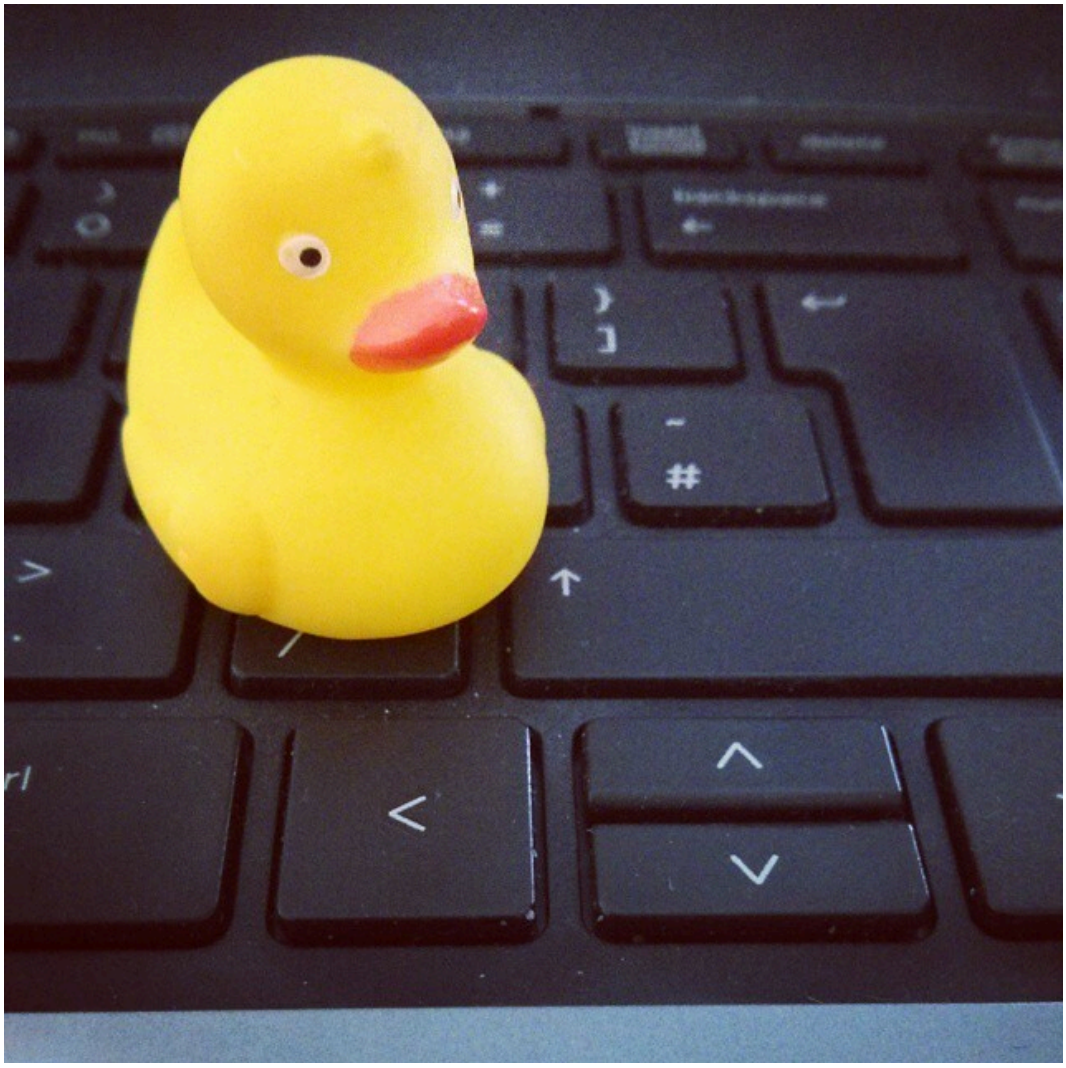
1. Écrire le nom du concept en haut d'une page blanche.
2. Lister le contexte du problème : les éléments connus, l'erreur rencontrée, les personnes ayant travaillé dernièrement sur le code.
3. Expliquer avec des phrases, comme si cela s'adressait à une autre personne, le problème et sa manifestation. Cela doit permettre de vous faire découvrir ce que vous ne savez pas sur le problème.
4. Explorer les zones d'ombres identifiées, poser des hypothèses et noter les différentes expériences que vous pouvez réaliser pour tester ces hypothèses.

5. Implémenter ces expériences et en fonction des résultats mettre à jour vos connaissances du problèmes en réitérant à partir de l'étape 3.

Méthode du canard en plastique (Rubber Duck Debugging)

Complément

En programmation, il existe une méthode similaire appelée « méthode du canard en plastique » : la méthode consiste à expliquer le problème à une personnage ou à une autre personne à haute voix. De manière similaire à la technique précédente, il est possible que la solution au problème vienne en explicitant celui-ci.



Debugging assistant #geek

Réaliser une correction incrémentale de l'exemple simplifié

Attention

Lorsque l'on corrige le problème, il est préférable d'utiliser une approche **incrémentale** et de ne modifier qu'un petit nombre d'instructions pour s'assurer que la correction que l'on opère est la bonne.

À retenir

Les méthodes pour la correction de bugs tiennent essentiellement en trois points :

- Une capacité à expliquer et donc comprendre le problème.
- Une capacité à modifier et simplifier du code pour isoler le problème.
- Une capacité à réaliser des recherches ciblées pour trouver de l'information.

V Exercice : Appliquer la notion

On veut afficher le résultat de toutes les multiplications des nombres entre 1 et 10.

Pour obtenir un résultat similaire à :

```
x   [1]  [2]  [3]  [4]  [5]  [6]  [7]  [8]  [9]  [10]
[1]   1   2   3   4   5   6   7   8   9   10
[2]   2   4   6   8   10   12   14   16   18   20
[3]   3   6   9   12   15   18   21   24   27   30
[4]   4   8   12   16   20   24   28   32   36   40
[5]   5   10   15   20   25   30   35   40   45   50
[6]   6   12   18   24   30   36   42   48   54   60
[7]   7   14   21   28   35   42   49   56   63   70
[8]   8   16   24   32   40   48   56   64   72   80
[9]   9   18   27   36   45   54   63   72   81   90
[10]   10   20   30   40   50   60   70   80   90   100
```

Pour cela, on a écrit le programme JavaScript suivant :

```
var result = 'x '
for (let i = 0; i < 11; i++) {
  for (let j = 0; j < 11; j++) {
    if (i === 0 || j > 0) {
      // Formattage pour l'en-tête
      result = result + '[' + j + ']'
    }
    if (j === 0 && i > 0) {
      // Formattage pour la première colonne
      result = result + '[' + i + '] '
    }
    if (i > 0 && j > 0) {
      // Intérieur du tableau
      result = result + (i * j) + ' '
    }
    result = result + '\t'
  }
  result = result + '\n'
}

console.log(result)
```

Question 1

[solution n°3 p. 35]

Exécuter ce script. Quel est le résultat ?

Question 2

[solution n°4 p. 36]

Au vu du problème précédent de formatage qui ajoute des crochets, on déduit que ce bug est nécessairement présent dans ce bloc :

```
if (i === 0 || j > 0) {
  // Formattage pour l'en-tête
  result = result + '[' + j + ']'
}
if (j === 0 && i > 0) {
  // Formattage pour la première colonne
  result = result + '[' + i + '] '
}
```

Afin de déboguer ce morceau de code on met en place cet autre morceau de code, qui supprime les boucles pour se concentrer sur le bug en lui-même.

```
const i = 4
const j = 3

result = ''


if(i == 0 || j > 0) {
  // Formattage pour l'en-tête
  result = result + '[' + j + ']'
}
if(j == 0 && i > 0) {
  // Formattage pour la première colonne
  result = result + '[' + i + '] '
}

console.log(result)
```

Exécuter ce bloc et trouver d'où provient le problème.

Question 3

[solution n°5 p. 36]

Corriger ce bloc et exécuter l'ensemble du code.

VI Outils pour la correction de bugs

Objectif

- Connaître certains outils pour corriger les bugs.

Mise en situation

On peut corriger les bugs avec les méthodes données précédemment, néanmoins il est aussi pratique de connaître les outils dédiés.

Interpréteur et console

Fondamental

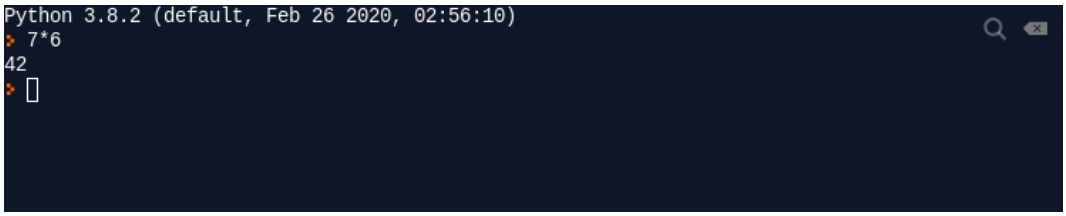
Afin de comprendre plus rapidement un problème, on peut utiliser un interpréteur (aussi appelé REPL) qui permet d'exécuter une instruction à la fois et d'obtenir le résultat dans la console.

Il est utile d'isoler une partie du code pour déboguer la partie comportant un problème.

Interpréteur Python sur Repl.it

Exemple

Dans Repl.it, le code que l'on écrit est exécuté dans un interpréteur Python, à droite dans l'interface. Néanmoins, on peut directement travailler avec cet interpréteur et saisir des instructions directement.



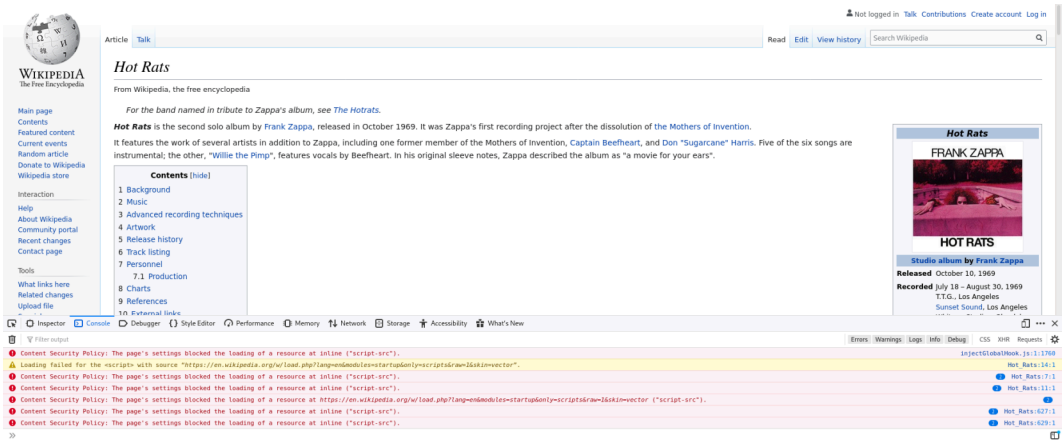
Interpréteur Python de Repl.it

Console Firefox

Exemple

La console permet d'obtenir les résultats retournés sur les sorties standard et d'erreurs : on obtient les valeurs données par `console.log` et les traces d'erreurs.

On y accède avec le raccourci `Ctrl + Shift + C`. On peut à partir de là utiliser l'interpréteur qui laisse entrer des commandes directement dans la fenêtre pour obtenir des résultats d'instructions.



Aperçu de la console sur un article Wikipédia. On voit qu'il existe des erreurs sur le script de cette page.

Outils de développement Firefox

Remarque

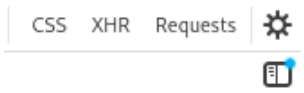
Lorsque l'on développe des applications web en JavaScript, on peut inspecter le rendu et l'exécution correcte du programme directement dans le navigateur.

Dans le cas de Firefox, on dispose de plusieurs outils pour réaliser cela.

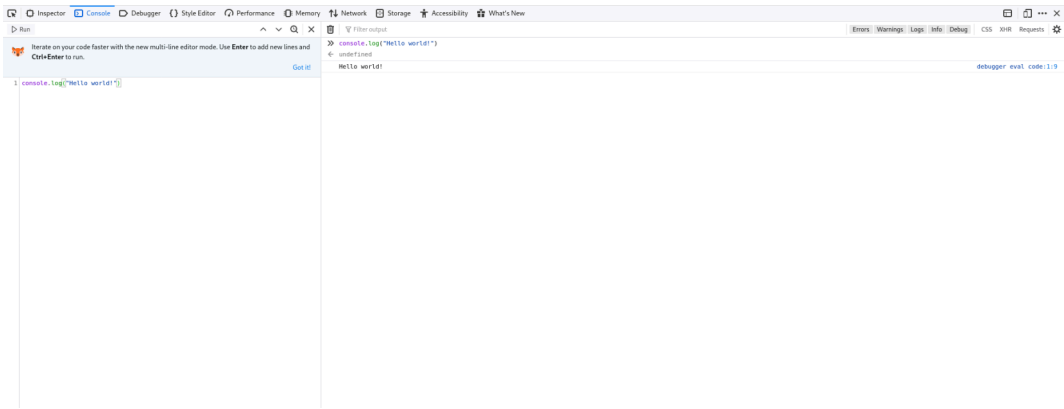
Approche incrémentale de développement ou de débogage

Méthode

Sous Firefox, on peut utiliser le mode d'édition multi-lignes pour avancer de manière incrémentale dans le développement du code ou dans le débogage. On l'active en cliquant sur :



On obtient une console en deux parties avec un éditeur et un interpréteur. On peut ainsi facilement modifier du code et l'exécuter ensuite.



Mode d'édition multi-lignes

Débogueur

Fondamental

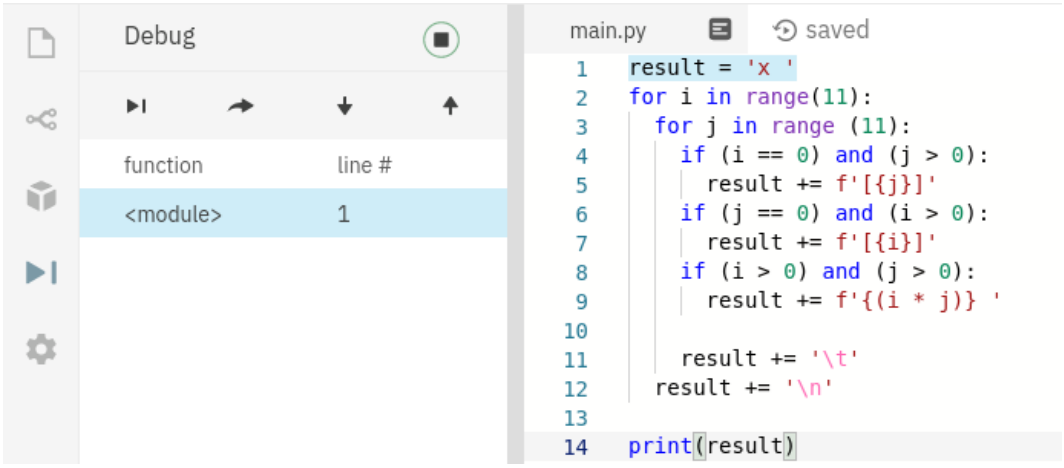
Dans le cas où l'on se trouve dans une structure de code plus complexe où l'on ne peut que difficilement isoler du code, on peut utiliser un débogueur.

Un débogueur permet d'inspecter l'entièreté de l'état du programme à partir de différentes lignes en marquant des points d'arrêt.

Débogueur Python de Repl.it

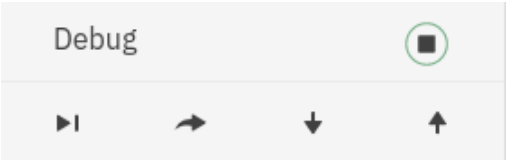
Exemple

Repl.it dispose d'un débogueur Python relativement simple qui permet de voir les différent appels de fonctions.



On peut entrer (*Step-Into*) dans l'exécution d'une instruction, en sortir (*Step-Out*), passer à l'instruction suivant (*Step-Over*) ou reprendre une exécution normale (*Resume*).

À noter que ce débogueur est en cours de développement et qu'il ne permet pas d'inspecter pour le moment l'entièreté de l'état du programme, comme les différentes valeurs des variables.

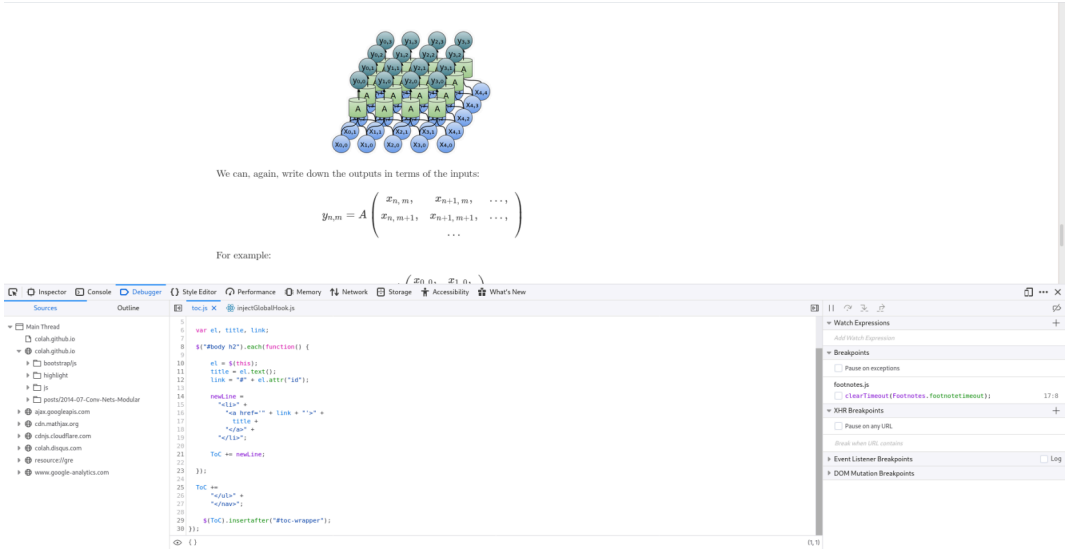


Débogueur JavaScript de Firefox

Exemple

Firefox dispose aussi d'un débogueur qui permet de jouer les lignes une à une à partir d'un point d'arrêt (*break-point*) ou lors d'une exception.

Comme dans le cas de la console, on y accède avec le raccourci `Ctrl + Shift + C`.

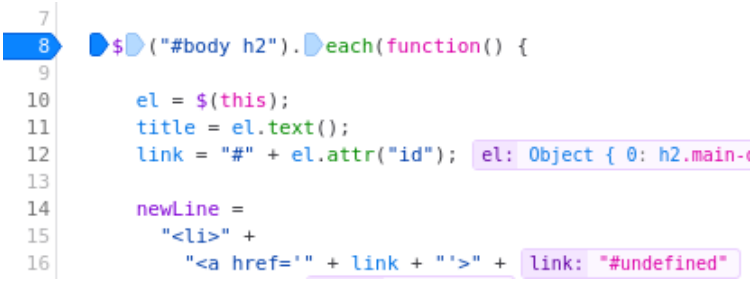


Aperçu du débogueur sur un article de blog (https://colah.github.io/)

Déboguer le code pas à pas avec le débogueur sous Firefox

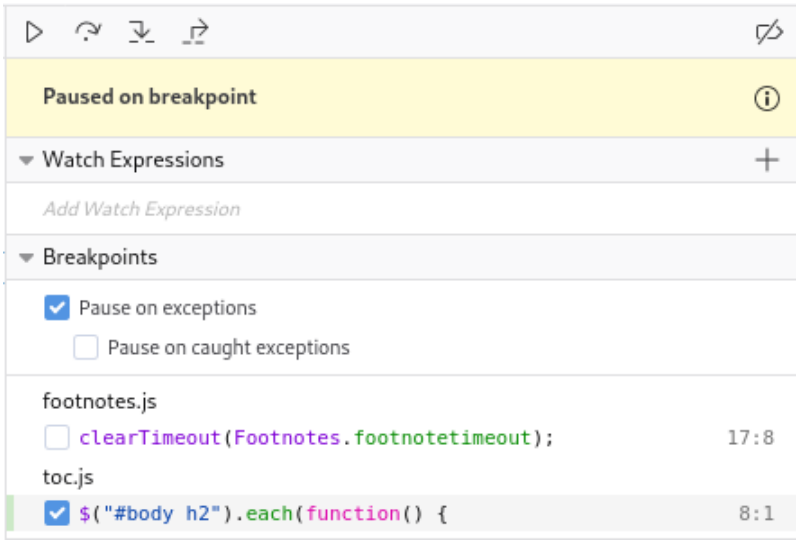
Méthode

Lorsque l'on tombe sur une erreur, on peut comprendre le contexte de celle-ci en parcourant le code à partir d'un endroit donné. Pour cela, on définit un point d'arrêt.



Point d'arrêt

Si on relance le code (en rafraîchissant la page), l'exécution de celui-ci s'arrête sur le point d'arrêt spécifié. On peut avoir un aperçu des différents points d'arrêts sélectionnés et on peut choisir d'avancer dans l'exécution du code instruction par instruction. Plus exactement, on peut choisir de passer à l'instruction suivante ou d'aller à l'intérieur de l'exécution du code d'une fonction.



Avancer dans le code

Pendant le débogage, on peut avoir accès à l'intégralité des valeurs des variables définies dans les différents *scopes*.

Parcours des scopes

On peut, en particulier, parcourir l'intégralité de l'arborescence des objets.

Documentation des outils de développement Firefox ⊕ Complément

On pourra consulter la documentation sur les outils de développement Firefox : https://developer.mozilla.org/fr/docs/Outils

À retenir

Il existe plusieurs outils pour développer ou inspecter un code en JavaScript : le débogueur, l'interpréteur (ou console).

Pour JavaScript les navigateurs web tels que Firefox disposent de leurs propres outils.

VII Exercice : Appliquer la notion

On veut formater l'adresse d'un lieu en France. On met en place le code suivant contenant les informations du lieu dans un objet JavaScript.

```
const addressInformation = {
  geometry: {
    coordinates: [
      2.29391,
      48.876318
    ],
    type: 'Point'
  },
  properties: {
    city: 'Paris',
    citycode: '75117',
    context: '75, Paris, Île-de-France',
    district: 'Paris 17e Arrondissement',
    id: '75117_0413',
    importance: 0.5812644699670191,
    label: "Rue de l'Arc de Triomphe 75017 Paris",
    name: "Rue de l'Arc de Triomphe",
    postcode: '75017',
    score: 0.6892058609060925,
    type: 'street',
    x: 648211.84,
    y: 6864264.35
  },
  type: 'Feature'
}

console.log('Ville : ' + addressInformation.city)
console.log('Contexte : ' + addressInformation.context)
console.log('Voie : ' + addressInformation.name)
console.log('Code Postal : ' + addressInformation.postcode)
```

Question 1

[solution n°6 p. 37]

Exécuter ce code dans la console Firefox en mode de saisie multi-lignes.

Qu'obtient-on comme résultat ? Celui-ci est-il correct ?

Question 2

[solution n°7 p. 37]

Inspecter la valeur de `addressInformation` dans la console de Firefox grâce à un point d'arrêt défini avant l'affichage.

Où se trouvent les champs `addressInformation.city`, `addressInformation.context`, `addressInformation.name`, et `addressInformation.postcode` dans l'objet `addressInformation` ?

Question 3

[solution n°8 p. 37]

Corriger le script en changeant l'accès aux composantes de `addressInformation` pour corriger le problème.

Exécuter le script obtenu. Quel affichage obtient-on ?

VIII Méthodes pour prévenir les bugs

Objectif

- Découvrir des méthodes et outils pour prévenir les bugs.

Mise en situation

Il est très important de corriger les bugs lorsqu'ils sont détectés, mais il est tout de même préférable de les empêcher d'arriver. Bien que le risque zéro n'existe pas, de nombreux outils et méthodes permettent de limiter au maximum les risques, et d'identifier un problème pendant les étapes de développement. Nous allons ici vous donner quelques techniques à suivre pour vous permettre de développer des programmes les plus fiables possibles. Certaines sont simples, d'autres plus compliquées à mettre en place, mais toutes sont utilisées couramment par les équipes de développement.

Mettre en place des tests

Fondamental

Lorsque l'on développe une nouvelle fonctionnalité pour un programme, on peut écrire des tests qui vont attester du bon fonctionnement de cette fonctionnalité lors de sollicitations attendues et inattendues. Lors de la correction d'un bug, on peut ajouter un test pour vérifier la validité du correctif.

Adopter les conventions de programmation

Fondamental

Se tenir à des conventions communes de programmation permet de structurer correctement le code et de permettre à de nouveaux programmeurs de ne pas réaliser des erreurs ensuite.

Convention de programmation pour JavaScript

Rappel

Il existe plusieurs standards de codage en JavaScript, qui se ressemblent. Il est d'usage d'en adopter un :

- w3schools.com[1]
- standardjs.com[2]

Ne pas dupliquer son code

Fondamental

Du code dupliqué introduit une maintenance plus complexe de celui-ci et des corrections incomplètes d'erreurs : on veillera à factoriser les parties de codes répétées.

Pour cela on utilise notamment des **fonctions**.

1. https://www.w3schools.com/js/js_conventions.asp
2. https://standardjs.com/rules.html

Utiliser des outils de développement

Fondamental

On peut utiliser des outils pour corriger son code :

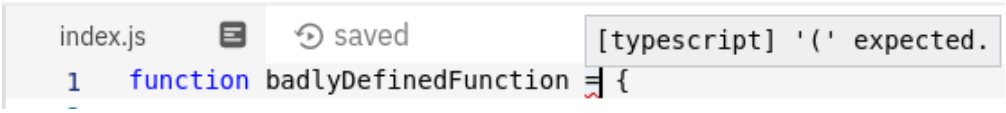
- les **linters** analysent la syntaxe de programme pendant qu'on l'écrit et qui indiquent les erreurs présentes au fur et à mesure ;
- les **formateurs** corrigent la forme du code produit pour que celui-ci respecte des standards.

Un linter permet uniquement de voir où se trouve un problème de syntaxe ou de format mais ne le corrige pas. Un formateur corrige un problème de formatage.

Linter JavaScript

Exemple

Repl.it dispose de son propre linter intégré dans son éditeur de texte : celui-ci indique les problèmes de syntaxe et de formatage.



Linter sous Repl.it indiquant une erreur

Il existe des extensions sur des éditeurs comme Atom (`linter-js-standard`) ou VSCode (`vscode-standardjs`) pour l'utiliser plus facilement lors du développement.

Formateurs JavaScript

Exemple

Repl.it dispose de son propre formateur accessible directement depuis son interface à l'aide d'un bouton.

Formateur de Repl.it

https://beautifier.io est un formateur JavaScript en ligne.

Il existe aussi des extensions sur des éditeurs comme Atom ou VSCode comme Atom Beautify (https://atom.io/packages/atom-beautify) pour utiliser directement des formateurs lors du développement.

Outils de différence de contenu de fichiers

Complément

On peut utiliser un outil de différence qui réalise la comparaison de chaînes de caractères entre deux fichiers afin de voir la différence avant et après formatage par exemple.

On peut pour cela utiliser un formateur en ligne comme https://www.diffchecker.com/ ou directement un utilitaire en ligne de commande comme `diff`.

À retenir

Les bugs peuvent se prévenir en adoptant de bonnes méthodes lors du développement.

Des outils de développement comme des linters ou des formateurs peuvent être utilisés pour appliquer plus facilement ces pratiques.

IX Exercice : Appliquer la notion

Dans le cadre de la gestion d'un planning de réunion qui regroupe des acteurs situés dans le monde entier, on choisit de traiter les dates qui apparaissent dans les échanges de mails.

Pour cela, on met un place un code qui formate une date à l'aide de l'interface `Date` de JavaScript pour afficher l'afficher dans un format explicite.

```
function formatDateToString( date ){

  const options = {
    weekday : 'long'
    year: 'numeric'
    month : 'long'
    day : 'numeric'
    hour : 'numeric'
    minute : 'numeric'
    second: 'numeric'
  }
  const locale = 'fr-FR'
  const dateString = new Date(date).toLocaleDateString(locale,options)

  return dateString
}

formatDateToString("2020-03-15")
```

Néanmoins, l'exemple précédent a des problèmes de syntaxe et est mal formaté.

Dans cet exercice, on va utiliser le linter de Repl.it et un formateur disponible en ligne (https://beautifier.io/) pour corriger ces problèmes.

Question 1

[solution n°9 p. 38]

Copier le contenu de ce fichier `index.js` dans Repl.it.

Question 2

[solution n°10 p. 38]

Le linter de Repl.it indique que le code comporte des erreurs, lesquelles ?

Corriger ces erreurs.

Question 3

[solution n°11 p. 39]

Formater le code corrigé avec en formateur en ligne comme https://beautifier.io/ ou directement avec le formateur de Repl.it.

Question 4

[solution n°12 p. 39]

Comparer les différences entre le code corrigé et non corrigé avec cet outil de différence en ligne : https://text-compare.com/

Quels sont les éléments qui ont été corrigés ?

X Essentiel

XI Quiz

Exercice 1 : Quiz - Culture

[solution n°13 p. 39]

Exercice

Quels sont les outils que l'on peut utiliser pour détecter et prévenir les bugs ?

- **A** Une console
- **B** Un débogueur
- **C** Un linter
- **D** Un formateur

Exercice

Quels sont les outils mis en place dans les navigateurs web pour aider à corriger les bugs JavaScript ?

- **A** Une console
- **B** Un débogueur
- **C** Un linter
- **D** Un formateur

Exercice 4 : Quiz - Méthode

[solution n°14 p. 40]

Exercice

Pourquoi veille-t-on à bien formater son code et à respecter des standards de codage ?

- **A** Pour que le code soit compatible avec les prochaines versions du langage de programmation utilisé.
- **B** Pour avoir un code lisible, ce qui permet à de nouveaux développeurs de mieux se le réapproprier.

Exercice

Utiliser les outils de développement et de formatage de code garantit de pouvoir corriger un bug.

- A Vrai
- B Faux

Exercice 7 : Quiz - Code

[solution n°15 p. 40]

Exercice

Quel type d'erreur contient ce programme ?

```
function numberOfDaysTillNextYearMarch13 {
  const now = new Date()
  const year = now.getFullYear()
  const march13 = new Date('13 March, ' + (year + 1))
  const numberOfMiliSeconds = march13.getTime() - now.getTime()
  return Math.floor(numberOfMiliSeconds / (1000 * 60 * 60 * 24))
}

console.log(numberOfDaysTillNextYearMarch13())

```

- A Une erreur de syntaxe
- B Une erreur de logique
- C Aucune erreur

Exercice

Quel type d'erreur contient ce programme ?

```
function numberOfDaysTillNextYearMarch13 () {
  const now = new Date()
  const year = now.getFullYear()
  const march13 = new Date('13 March, ' + (year + 1))
  const numberOfMiliSeconds = march13.getTime() - now.getTime()
  return Math.floor(numberOfMiliSeconds / (1000 * 60 * 24))
}

console.log(numberOfDaysTillNextYearMarch13())

```

- A Une erreur de syntaxe lors de la définition de fonction
- B Une erreur de logique
- C Aucune erreur

Exercice

Quel type d'erreur contient ce programme ?

```
function numberOfDaysTillNextYearMarch13 () {
  const now = new Date()
  const year = now.getFullYear()
  const march13 = new Date('13 March, ' + (year + 1))
  const numberOfMiliSeconds = march13.getTime() - now.getTime()
  return Math.floor(numberOfMiliSeconds / (1000 * 60 * 60 * 24))
}

console.log(numberOfDaysTillNextYearMarch13())

```

- **A** Une erreur de syntaxe lors de la définition de fonction
- **B** Une erreur de logique
- **C** Aucune erreur

XII Exercice : Défi

On veut calculer le prix total d'une commande de plusieurs articles réalisée sur un site web de papeterie. On dispose des informations de la commande sous forme d'un tableau de cette forme :

Article	Prix Unitaire Hors Taxes (en €)	Quantité achetée	Taux TVA
Carnet Capiha Format A6	4.32 €	2	5 %
Cartouche d'encre bleu Water Melon Man	2.71 €	5	20 %
Stylo plume Carpeur	26.82 €	1	20 %

Pour réaliser la commande il faut calculer :

- le montant hors-taxe de la commande ;
- appliquer une majoration pour trouver le prix TTC (« Toutes taxes comprises ») de la commande.

Question 1

[solution n°16 p. 42]

Calculer à la main le montant hors-taxe du panier.

Indice :

Le montant hors-taxe est égal à la somme de tous les prix unitaires hors-taxe, multipliés par la quantité achetée.

Question 2

[solution n°17 p. 42]

On se donne le morceau de code suivant reprenant le tableau donné plus haut et la logique pour calculer le prix hors taxe du panier.

```
const productsTable =
[['Carnet Capiha Format A6', 4.32, 2, 0.05],
["Cartouche d'encre bleu Water Melon Man", 2.71, 5, 0.2],
['Stylo plume Carpeur', 26.82, 1, 0.2]]


let dutyFreePrice = 0
for (let line of productsTable) {
    dutyFreePrice = dutyFreePrice + line[1] + line[2]
}

console.log('Total Price: ', dutyFreePrice, ' €')
```

Quel est la sortie retournée par le programme ?

Le résultat est-il correct ? Corriger s'il faut l'erreur et vérifier que le résultat est bien bon.

Question 3

[solution n°18 p. 42]

Exécuter le script corrigé.

Quelle est la sortie du programme ? Le comportement de ce nouveau script corrigé est-il correct ?

Question 4

[solution n°19 p. 42]

Calculer à la main le montant TTC du panier.

Indice :

Le montant TTC pour chaque ligne de produit est égal à son prix hors-taxe, multiplié par `(1 - pourcentageTVA)`.

Question 5

[solution n°20 p. 42]

On veut maintenant calculer le montant du panier avec TVA dans une variable.

On se donne le morceau de code suivant reprenant le tableau donné plus haut et la logique pour calculer le prix du panier après application de la TVA.

```
1  const productsTable =
2  [['Carnet Capiha Format A6', 4.32, 2, 0.05],
3  ["Cartouche d'encre bleu Water Melon Man", 2.71, 5, 0.2],
4  ['Stylo plume Carpeur', 26.82, 1, 0.2]]
5
6
7  let price = 0
8  for (let line of productsTable) {
9      price = price + line[1] * line[2]
10     price = price * line[3]
11 }
12
13 console.log('Total Price with taxe: ', price.toFixed(2), ' €')
```

Quel est la sortie retournée par le programme ?

Le résultat est-il correct ? Corriger s'il faut l'erreur et vérifier que le résultat est bien bon.

Question 6

[solution n°21 p. 43]

On veut maintenant calculer et afficher comme précédemment le prix avec TVA de plusieurs paniers donnés dans une liste.

```
1  const productsTables = [
2
3    [['Cahier Sombreroche', 7.00, 4, 0.05],
4    ['Feutre indélébile générique', 3.99, 5, 0.2],
5    ['Stylo à bille Sib', 35, 12, 0.2]],
6
7    [['Carnet Capiha Format A4', 8.23, 1, 0.05],
8    ["Cartouche de feuille d'imprimante 90g/m²", 5, 2, 0.2]],
9
10   [['Lot de craies', 3.99, 3, 0.2],
11   ['Tableau blanc', 12.00, 1, 0.2]]
12
13 ]
```

On met en place naïvement le code suivant :

```
const productsTables = [

  [['Cahier Sombreroche', 7.00, 4, 0.05],
  ['Feutre indélébile générique', 3.99, 5, 0.2],
  ['Stylo à bille Sib', 35, 12, 0.2]],

  [['Carnet Capiha Format A4', 8.23, 1, 0.05],
  ["Cartouche de feuille d'imprimante 90g/m²", 5, 2, 0.2]],

  [['Lot de craies', 3.99, 3, 0.2],
  ['Tableau blanc', 12.00, 1, 0.2]]

]

let price = 0
for (let line of productsTables[0]) {
    price = price + line[1] * line[2] * (1 + line[3])
}

console.log('Total Price with taxe: ', price.toFixed(2), ' €')

price = 0
for (let line of productsTables[1]) {
    price = price + line[1] * line[2] * (1 + line[3])
}

console.log('Total Price with taxe: ', price.toFixed(2), ' €')


price = 0
for (let line of productsTables[1]) {
    price = price + line[1] * line[2] * (1 + line[3])
}

console.log('Total Price with taxe: ', price.toFixed(2), ' €')
```

Exécuter ce code. Il y a-t-il une erreur ? Comment la corriger ?

Question 7

[solution n°22 p. 43]

Après correction, le code précédent contient de la duplication de code, ce qui est une très mauvaise pratique.

Proposer un moyen qui permette à la fois de réduire ce code, d'éviter des bugs et de gagner en compréhension, tout en gardant le même résultat.

Indice :

On peut pour cela injecter la logique dans une fonction dédiée au calcul du montant avec taxe d'un panier.

Question 8

[solution n°23 p. 44]

Donner le rendu final du code une fois formaté :

Indice :

On peut utiliser un formateur en ligne comme : https://beautifier.io/

Conclusion

Les bugs sont des erreurs, bien connues du grand public, qui surviennent lors de l'exécution d'un programme informatique. Le travail d'un développeur est de limiter au maximum les bugs dans son programme, mais aussi de les corriger lorsqu'ils sont détectés. Pour cela il peut s'appuyer sur différentes méthodes et outils, largement répandus. En entreprise, la plupart des équipes de développement mettent ainsi en place un ensemble de règles et de tests à suivre lors du développement, pour assurer la meilleure fiabilité possible.

Solutions des exercices

Solution n°1

[exercice p. 7]

C'est un bug de syntaxe : il faut ajouter des parenthèses sur la condition.

```
1 const i = 2
2 if (i % 2) {
3   console.log(i + ' est impair')
4 }
```

Solution n°2

[exercice p. 7]

Il s'agit d'une erreur de logique. Ici, si on exécute, on obtient :

```
1 1 est pair
```

Il faut changer le message :

```
1 const i = 1
2 if (i % 2 === 0) {
3   console.log(i + ' est pair')
4 }
```

Ou bien changer la condition :

```
1 const i = 1
2 if (i % 2 !== 0) {
3   console.log(i + ' est impair')
4 }
```

Solution n°3

[exercice p. 15]

On n'obtient pas d'erreur mais on obtient un mauvais résultat :

```
1 x [0]  [1]  [2]  [3]  [4]  [5]  [6]  [7]  [8]  [9]  [10]
2 [1]   [1]1   [2]2   [3]3   [4]4   [5]5   [6]6   [7]7   [8]8   [9]9   [10]10
3 [2]   [1]2   [2]4   [3]6   [4]8   [5]10   [6]12   [7]14   [8]16   [9]18   [10]20
4 [3]   [1]3   [2]6   [3]9   [4]12   [5]15   [6]18   [7]21   [8]24   [9]27   [10]30
5 [4]   [1]4   [2]8   [3]12   [4]16   [5]20   [6]24   [7]28   [8]32   [9]36   [10]40
6 [5]   [1]5   [2]10   [3]15   [4]20   [5]25   [6]30   [7]35   [8]40   [9]45   [10]50
7 [6]   [1]6   [2]12   [3]18   [4]24   [5]30   [6]36   [7]42   [8]48   [9]54   [10]60
8 [7]   [1]7   [2]14   [3]21   [4]28   [5]35   [6]42   [7]49   [8]56   [9]63   [10]70
9 [8]   [1]8   [2]16   [3]24   [4]32   [5]40   [6]48   [7]56   [8]64   [9]72   [10]80
10 [9]   [1]9   [2]18   [3]27   [4]36   [5]45   [6]54   [7]63   [8]72   [9]81   [10]90
11 [10]   [1]10   [2]20   [3]30   [4]40   [5]50   [6]60   [7]70   [8]80   [9]90
   [10]100
```

Il y a un problème de logique dans ce programme qui affiche des nombres en trop entre crochets.

Solution n°4

[exercice p. 15]

On obtient le résultat suivant :

```
[3]
```

Or dans la situation suivante, nous devrions être à l'intérieur du tableau et il ne devrait donc ne pas y avoir un tel affichage avec un crochet.

Du fait des valeurs de `i` et `j`, on en déduit que c'est la valeur de `j` qui est affichée et ainsi que le test a un problème. En effet, il faut nécessairement que `i` soit nul et que `j` soit positif strictement. Ainsi, on déduit que le premier test est incorrect.

Solution n°5

[exercice p. 16]

Il faut corriger le code change le test pour vérifier que `i` soit nul et que `j` soit positif :

```
if(i === 0 && j > 0){
  // Formattage pour l'en-tête
  result = result + '[' + j + ']'
}
```

L'ensemble du code est :

```
var result = 'x '
for (let i = 0; i < 11; i++) {
  for (let j = 0; j < 11; j++) {
    if (i === 0 && j > 0) {
      // Formattage pour l'en-tête
      result = result + '[' + j + ']'
    }
    if (j === 0 && i > 0) {
      // Formattage pour la première colonne
      result = result + '[' + i + '] '
    }
    if (i > 0 && j > 0) {
      // Intérieur du tableau
      result = result + (i * j) + ' '
    }
    result = result + '\t'
  }
  result = result + '\n'
}

console.log(result)
```

On obtient bien le résultat souhaité, à savoir :

```
x   [1]  [2]  [3]  [4]  [5]  [6]  [7]  [8]  [9]  [10]
[1]   1   2   3   4   5   6   7   8   9   10
[2]   2   4   6   8   10   12   14   16   18   20
[3]   3   6   9   12   15   18   21   24   27   30
[4]   4   8   12   16   20   24   28   32   36   40
[5]   5   10   15   20   25   30   35   40   45   50
[6]   6   12   18   24   30   36   42   48   54   60
[7]   7   14   21   28   35   42   49   56   63   70
[8]   8   16   24   32   40   48   56   64   72   80
[9]   9   18   27   36   45   54   63   72   81   90
[10]   10   20   30   40   50   60   70   80   90   100
```

Solution n°6

[exercice p. 22]

On obtient le résultat suivant :

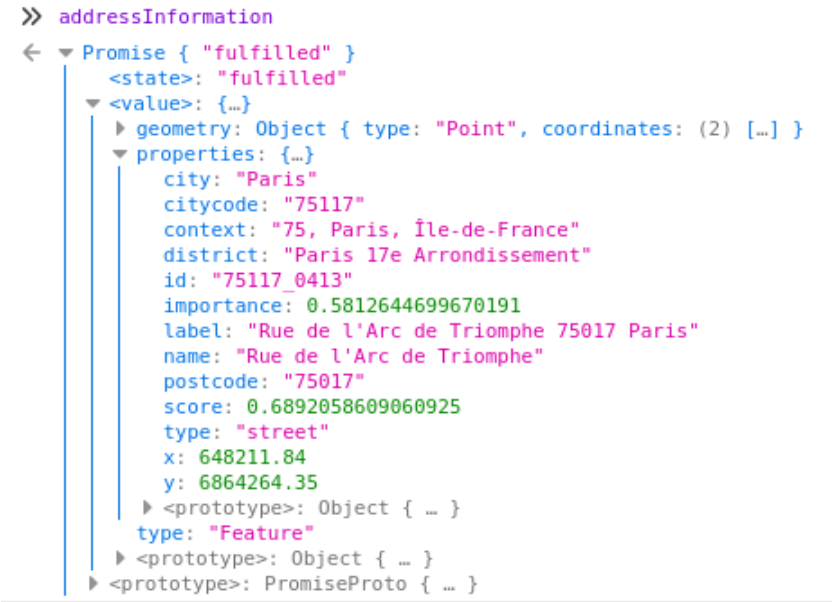
```
Ville : undefined
Contexte : undefined
Voie : undefined
Code Postal : undefined
```

Ce résultat n'est pas correct : il semblerait que les champs `addressInformation.city`, `addressInformation.context`, `addressInformation.name` et `addressInformation.postcode` ne soient pas définis.

Solution n°7

[exercice p. 22]

On inspecte la valeur d'`addressInformation` ainsi :



Ces champs se trouvent dans l'attribut `properties` de `addressInformation`.

Solution n°8

[exercice p. 22]

On modifie le code final pour faire apparaître `properties`. Le code final est :

```
const addressInformation = {
  geometry: {
    coordinates: [
      2.29391,
      48.876318
    ],
    type: 'Point'
  },
  properties: {
    city: 'Paris',
    citycode: '75117',
    context: '75, Paris, Île-de-France',
    district: 'Paris 17e Arrondissement',
    id: '75117_0413',
    importance: 0.5812644699670191,
    label: "Rue de l'Arc de Triomphe 75017 Paris",
    name: "Rue de l'Arc de Triomphe",
    postcode: '75017',
```

```
    score: 0.6892058609060925,
    type: 'street',
    x: 648211.84,
    y: 6864264.35
  },
  type: 'Feature'
}

console.log("Ville : " + addressInformation.properties.city)
console.log("Contexte : " + addressInformation.properties.context)
console.log("Voie : " + addressInformation.properties.name)
console.log("Code Postal : " + addressInformation.properties.postcode)
```

On obtient :

```
Ville : Paris
Contexte : 75, Paris, Île-de-France
Voie : Rue de l'Arc de Triomphe
Code Postal : 75017
```

Solution n°9

[exercice p. 26]

Solution n°10

[exercice p. 26]

On obtient des erreurs du type :

```
    year: 'numeric'
    ^^^^

SyntaxError: Unexpected identifier
```

En effet, un enregistrement doit séparer ses composantes par des virgules, qu'il faut ajouter dans `options`.

```
function formatDateToString( date ){

  const options = {
    weekday : 'long',
    year: 'numeric',
    month : 'long',
    day : 'numeric',
    hour : 'numeric',
    minute : 'numeric',
    second: 'numeric'
  }
  const locale = 'fr-FR'
  const dateString = new Date(date).toLocaleDateString(locale,options)

  return dateString
}

formatDateToString("2020-03-15")
```

Solution n°11

[exercice p. 26]

On obtient le code formaté suivant avec https://beautifier.io/ :

```
function formatDateToString(date) {

    const options = {
        weekday: 'long',
        year: 'numeric',
        month: 'long',
        day: 'numeric',
        hour: 'numeric',
        minute: 'numeric',
        second: 'numeric'
    }
    const locale = 'fr-FR'
    const dateString = new Date(date).toLocaleDateString(locale, options)

    return dateString
}

formatDateToString("2020-03-15")
```

Solution n°12

[exercice p. 26]

Les indentations et les espaces ont été corrigés.

Solution n°13

[exercice p. 28]

Exercice

Quels sont les outils que l'on peut utiliser pour détecter et prévenir les bugs ?

A Une console

B Un débogueur

C Un linter

D Un formateur

- Les **linters** indiquent les erreurs présentes au fur et à mesure.
- Les **formateurs** corrigent la forme du code produit pour qu'il soit lisible.

Exercice

Quels sont les outils mis en place dans les navigateurs web pour aider à corriger les bugs JavaScript ?

A Une console

B Un débogueur

C Un linter

D Un formateur

Solution n°14

[exercice p. 28]

Exercice

Pourquoi veille-t-on à bien formater son code et à respecter des standards de codage ?

A
Pour que le code soit compatible avec les prochaines versions du langage de programmation utilisé.

B
Pour avoir un code lisible, ce qui permet à de nouveaux développeurs de mieux se le réapproprier.

Exercice

Utiliser les outils de développement et de formatage de code garantit de pouvoir corriger un bug.

A Vrai

B Faux

S'il y a une erreur de logique dans le programme, il y aura un ou plusieurs bugs sans qu'aucun outil ne puisse corriger le problème automatiquement.

Solution n°15

[exercice p. 29]

Exercice

Quel type d'erreur contient ce programme ?

```
1 function numberOfDaysTillNextYearMarch13 {
2   const now = new Date()
3   const year = now.getFullYear()
4   const march13 = new Date('13 March, ' + (year + 1))
5   const numberOfMiliSeconds = march13.getTime() - now.getTime()
6   return Math.floor(numberOfMiliSeconds / (1000 * 60 * 60 * 24))
7 }
8
9 console.log(numberOfDaysTillNextYearMarch13())
10
```

A Une erreur de syntaxe

B Une erreur de logique

C Aucune erreur

Il y a une erreur de syntaxe lors de la définition de fonction : il faut ajouter des parenthèses après la définition de la fonction.

Exercice

Quel type d'erreur contient ce programme ?

```
function numberOfDaysTillNextYearMarch13 () {
  const now = new Date()
  const year = now.getFullYear()
  const march13 = new Date('13 March, ' + (year + 1))
  const numberOfMiliSeconds = march13.getTime() - now.getTime()
  return Math.floor(numberOfMiliSeconds / (1000 * 60 * 24))
}

console.log(numberOfDaysTillNextYearMarch13())

```

A Une erreur de syntaxe lors de la définition de fonction

B Une erreur de logique

C Aucune erreur

Il y a une erreur de logique : la dernière conversion n'est pas correcte.

Exercice

Quel type d'erreur contient ce programme ?

```
function numberOfDaysTillNextYearMarch13 () {
  const now = new Date()
  const year = now.getFullYear()
  const march13 = new Date('13 March, ' + (year + 1))
  const numberOfMiliSeconds = march13.getTime() - now.getTime()
  return Math.floor(numberOfMiliSeconds / (1000 * 60 * 60 * 24))
}

console.log(numberOfDaysTillNextYearMarch13())

```

A Une erreur de syntaxe lors de la définition de fonction

B Une erreur de logique

C Aucune erreur

Solution n°16

[exercice p. 31]

Le montant hors taxe réel est : 4.32 × 2 + 2.71 × 5 + 26.82 × 1 = 49.01 €.

Solution n°17

[exercice p. 31]

La sortie retournée est :

```
Total Price:  41.85  €
```

Le résultat est différent de celui qu'on a calculé à la main. Il y a un problème de logique dans le script, il faut remplacer la somme par une multiplication.

```
dutyFreePrice = dutyFreePrice + line[1] * line[2]
```

On obtient :

```
const productsTable =
[['Carnet Capiha Format A6', 4.32, 2, 0.05],
["Cartouche d'encre bleu Water Melon Man", 2.71, 5, 0.2],
['Stylo plume Carpeur', 26.82, 1, 0.2]]


let dutyFreePrice = 0
for (let line of productsTable) {
    dutyFreePrice = dutyFreePrice + line[1] * line[2]
}

console.log('Total Price: ', dutyFreePrice.toFixed(2), ' €')
```

Solution n°18

[exercice p. 32]

Le résultat que l'on obtient est bien correct.

```
Total Price:  49.01  €
```

Solution n°19

[exercice p. 32]

Le coût réel du panier est : 4.32 × 2 × 1.05 + 2.71 × 5 × 1.2 + 26.82 × 1 × 1.2 = 57.52 €.

Solution n°20

[exercice p. 32]

Le résultat est mauvais, on obtient :

```
Total Price with taxe:  5.92328  €
```

Il y a des erreurs sur les lignes de calculs, on corrige cela pour :

```
for (let line of productsTable) {
    price = price + line[1] * line[2] * (1 + line[3])
}
```

On obtient le script final :

```
const productsTable =
[['Carnet Capiha Format A6', 4.32, 2, 0.05],
["Cartouche d'encre bleu Water Melon Man", 2.71, 5, 0.2],
['Stylo plume Carpeur', 26.82, 1, 0.2]]


let price = 0
for (let line of productsTable) {
    price = price + line[1] * line[2] * (1 + line[3])
}

console.log('Total Price with taxe: ', price.toFixed(2), ' €')
```

On obtient bien :

```
Total Price with taxe:  57.52  €
```

Solution n°21

[exercice p. 32]

Oui, il y a une erreur : le prix du second panier est affiché à la place de celui du troisième panier.

On doit normalement obtenir 28.76 €.

La dernière boucle devient :

```
for (let line of productsTables[2])
```

La bonne valeur est alors affichée.

Solution n°22

[exercice p. 33]

```
const productsTables = [

  [['Cahier Sombreroche', 7.00, 4, 0.05],
  ['Feutre indélébile générique', 3.99, 5, 0.2],
  ['Stylo à bille Sib', 35, 12, 0.2]],

  [['Carnet Capiha Format A4', 8.23, 1, 0.05],
  ["Cartouche de feuille d'imprimante 90g/m²", 5, 2, 0.2]],

  [['Lot de craies', 3.99, 3, 0.2],
  ['Tableau blanc', 12.00, 1, 0.2]]

]

function computeTaxesProducts(productsTable) {
    let price = 0
    for (let line of productsTable) {
        price = price + line[1] * line[2] * (1 + line[3])
    }
    return price.toFixed(2)
}

for (let table of productsTables) {
    console.log('Total Price with taxe: ', computeTaxesProducts(table))
}

```

On obtient :

```
Total Price with taxe:  557.34  €
Total Price with taxe:  20.64  €
Total Price with taxe:  28.76  €
```

Solution n°23

[exercice p. 33]

```
const productsTables = [

    [
        ['Cahier Sombreroche', 7.00, 4, 0.05],
        ['Feutre indélébile générique', 3.99, 5, 0.2],
        ['Stylo à bille Sib', 35, 12, 0.2]
    ],

    [
        ['Carnet Capiha Format A4', 8.23, 1, 0.05],
        ["Cartouche de feuille d'imprimante 90g/m²", 5, 2, 0.2]
    ],

    [
        ['Lot de craies', 3.99, 3, 0.2],
        ['Tableau blanc', 12.00, 1, 0.2]
    ]

]

function computeTaxesProducts(productsTable) {
    let price = 0
    for (line of productsTable) {
        price = price + line[1] * line[2] * (1 + line[3])
    }
    console.log('Total Price with taxe: ', price.toFixed(2), ' €')
}

for (let table of productsTables) {
    computeTaxesProducts(table)
}
```

La lisibilité, notamment au niveau de la déclaration des différents paniers, est améliorée.

Crédits des ressources

Premier cas de bogue avéré p. 6

Licence : Domaine Public - Courtesy of the Naval Surface Warfare Center, Dahlgren, VA., 1988. (Wikipédia, Article Bug (informatique) : https://fr.wikipedia.org/wiki/Bug_(informatique))

Debugging assistant #geek p. 13

Attribution - Partage dans les Mêmes Conditions - Leonid Mamchenkov (https://www.flickr.com/photos/mamchenkov/16723393075)