Vk

Zeste de savoir

Deboguer ses macros VBA dans Excel

mardi 02 juillet 2024

Table des matières

Introduction

Comme toute suite d'instructions, les macros VBA n'échappent pas aux dures erreurs de la programmation.

Parmi ces erreurs, nous pouvons remarquer que certaines sont liées à la syntaxe (ce qui empêche le programme de comprendre l'instruction) ou encore que d'autres sont liées à la logique (ce qui fait que le programme plante ou donne des résultats erronés). S'il est en général facile de se prémunir et de corriger les premières, les secondes peuvent vite donner un peu plus de fil à retordre ! 🎃

Au cours de ce billet, nous allons voir comment tirer profit de l'éditeur VBE et du VBA pour éviter ces désagréments et pour déboguer nos macros.

C'est parti !

1. Se prémunir des erreurs

Les erreurs ne sont pas fatidiques, mais pas inévitables non plus. Quoiqu'il en soit, il existe différentes approches pour nous en prémunir le plus possible et nous allons en étudier quelques unes durant cette première section.

1.1. Se prémunir des erreurs de syntaxe

Les erreurs liées à la syntaxe sont de légers problèmes dus bien souvent à un manque d'attention ou à une méconnaissance du langage.

```
1 Debug.Print ("Hello World!")
2 Debug.Print "Hello", "World!"
3 'Debug.Print("Hello", "World!") 'Erreur de syntaxe
```

L'éditeur met en avant le code problématique afin de nous alerter qu'il y a quelque chose qui cloche :

```
Debug.Print ("Hello World!")
Debug.Print "Hello", "World!"
Debug.Print("Hello", "World!") 'Erreur de syntaxe
```

FIGURE 1.1. – Mise en rouge de l'erreur

Si nous souhaitons avoir plus de précisions sur l'erreur, nous devons faire apparaître cette petite fenêtre :

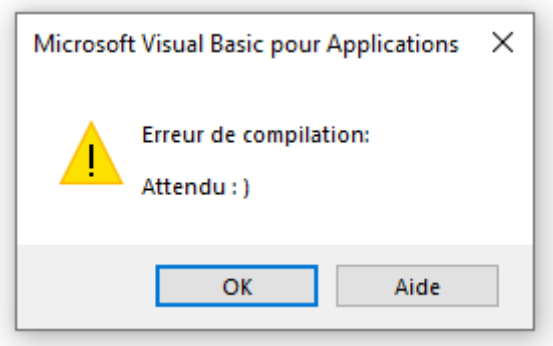

FIGURE 1.2. – Erreur compilation

Dans celle-ci, se trouvent un message plus ou moins parlant ainsi qu'un bouton "Aide" qui, dans notre cas, nous redirigera vers cette page où nous pouvons apprendre dans le point "Expected :)" qu'il ne faut pas mettre de parenthèses ici.

La traduction française de la documentation n'est pas toujours très compréhensive, c'est pourquoi je vous recommande de passer en anglais lorsque c'est possible et que ce n'est pas moins compréhensif pour vous en changeant le "/fr-fr/" en "/en-us/" dans l'URL.

Pour faire apparaître ces petites fenêtres, il y a plusieurs possibilités.

1.1.1. Vérification automatique de la syntaxe

Tout d'abord, nous pouvons activer l'option "Vérification automatique de la syntaxe" de la fenêtre "Options" accessible via le menu "Outils" de l'éditeur.

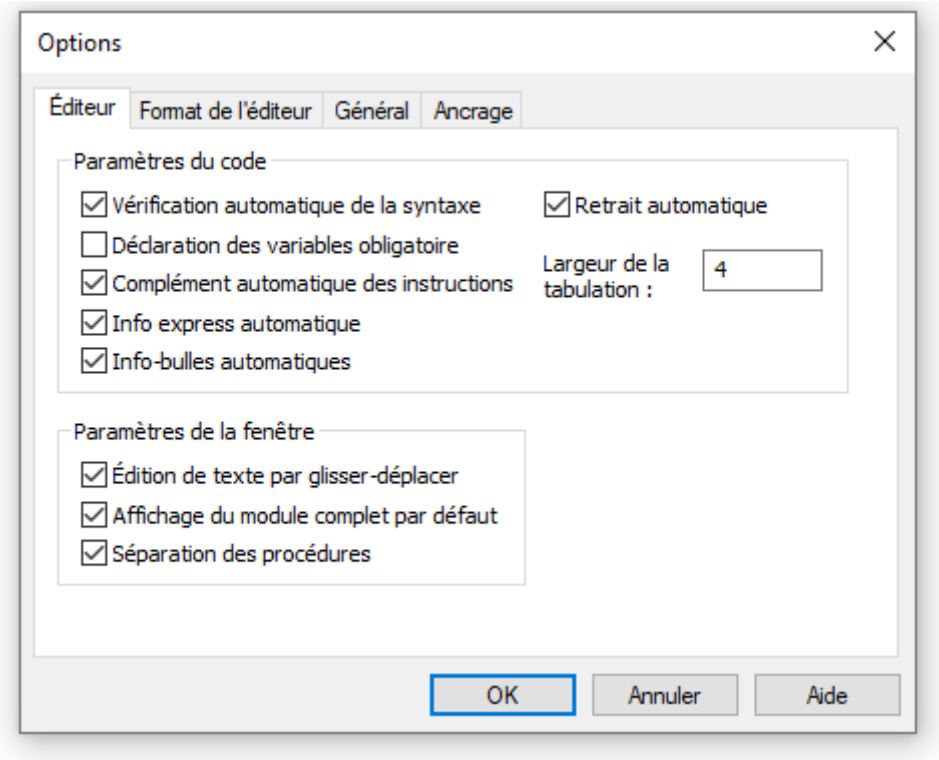

FIGURE 1.3. – Fenêtre "Options" VBE

En cochant celle-ci, la fenêtre d'erreur s'affichera directement une fois le code erroné écrit.

C'est une option intéressante pour débuter, mais qui peut devenir un peu énervante avec de l'expérience, car la fenêtre est assez intempestive et la coloration de l'erreur suffit à nous alerter.

1.1.2. Compilation du projet

Un autre moyen de vérifier son code et d'afficher les erreurs est de compiler le projet. Cela se fait via le menu "Débogage".

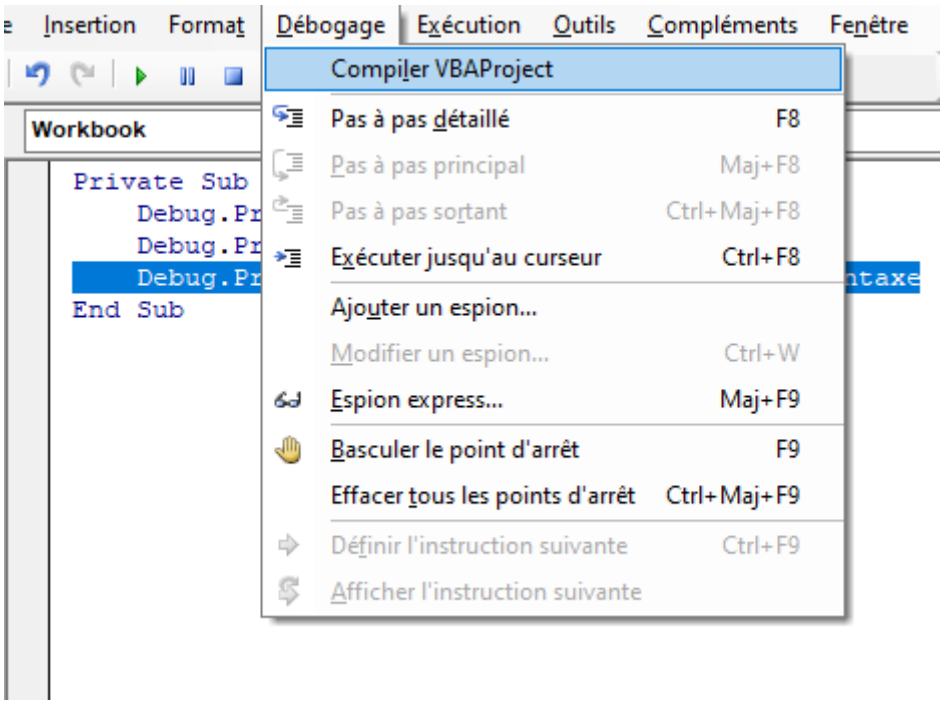

FIGURE 1.4. – Compiler VBAProject

Cette solution est moins dérangeante. Néanmoins la compilation s'arrêtant à la première erreur trouvée, il peut être judicieux de la lancer régulièrement pour éviter de tout rattraper d'un coup.

1.2. Se prémunir des erreurs de logique

Les erreurs liées à la logique peuvent être causées par différentes choses (boucle incorrecte, variable prenant une mauvaise valeur, etc.). Elles peuvent amener en conséquence des erreurs à l'exécution ou un résultat erroné mais sans erreur d'exécution.

Là encore, nous pouvons noter quelques façons de se prémunir de ces erreurs.

1.2.1. Déclarer ses variables

Par défaut, il n'est pas obligatoire de déclarer ses variables. Toutefois, il est recommandé de le faire. En plus d'obtenir un gain de clarté et de performance, cela évite des problèmes induits par des erreurs de saisie.

```
1 iNombre1 = 5
2 iNombre2 = 10
3
4 Debug.Print ("Résultat : " & iNombre1 + iNombre3) ' Résultat : 5
```

Pour être sûr de ne pas oublier une déclaration, nous pouvons ajouter `Option Explicit` en début de module. Cela aura pour effet de lever une erreur en de cas variable non déclarée :

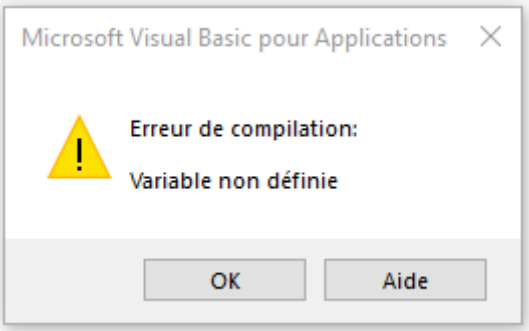

FIGURE 1.5. – Variable non définie

Remarquons qu'en cochant l'option "Déclaration des variables obligatoire" de la fenêtre "Options" vue précédemment, ce code est automatiquement ajouté dans les nouveaux modules.

1.2.2. Concevoir son programme

Concevoir son programme ou son algorithme en amont en langage naturel peut aider à l'établir avec clarté pour le transcrire ensuite en langage informatique, réduisant ainsi les possibles erreurs de logique.

1.2.3. Tester

Enfin, il peut être bienvenue d'effectuer des tests manuels ou automatisés pour se convaincre que le code a effectivement le comportement attendu.

Au fil de cette section, nous avons vu comment éviter et corriger certaines erreurs. Certaines erreurs seront plus coriaces et il faudra lors chercher à comprendre leur origine.

2. Analyser l'évolution des données...

Durant l'exécution du programme, les données vont vivre. Elles vont prendre une valeur, puis potentiellement évoluer au fil des opérations.

Si elles n'ont pas la valeur que nous pensons qu'elles doivent avoir, c'est que quelque chose ne va pas. Il faut alors comprendre ce qui ne fonctionne pas correctement.

Il y a plusieurs façons de suivre ces données.

2.1. ...en les écrivant

Nous pouvons les écrire quelque part.

2.1.1. Dans une boîte de dialogue avec MsgBox

La fonction `MsgBox` permet d'ouvrir une fenêtre de dialogue avec le texte voulu.

```
1 Dim iNombre1 As Integer
2 Dim iNombre2 As Integer
3 iNombre1 = 5
4 iNombre2 = 10
5
6 MsgBox "Nombre 1 = " & iNombre1 & vbNewLine & _
7        "Nombre 2 = " & iNombre2 & vbNewLine & _
8        "Résultat = " & iNombre1 + iNombre2
```

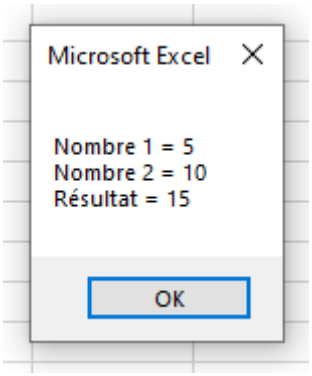

FIGURE 2.6. – Utilisation d'une boîte de dialogue

L'avantage de cette approche est de bloquer l'exécution le temps que la fenêtre est ouverte. Par contre, elle n'est pas adaptée s'il faut suivre beaucoup d'informations comme dans une boucle par exemple.

2.1.2. Dans la fenêtre d'exécution avec Debug.Print

La méthode `Print` de l'objet `Debug` permet d'écrire du texte dans la fenêtre d'exécution.

i Si la fenêtre d'exécution n'est pas visible, il faut l'afficher via le sous-menu "Fenêtre Exécution" dans le menu "Affichage" (raccourci Ctrl + G)

C'est l'approche que nous avons suivie jusqu'à présent et que j'utilise principalement. Elle a l'avantage d'être simple et non bloquante. Le fait qu'elle soit non bloquante peut être vu comme un inconvénient, car on peut se noyer dans la masse de données à moins d'écrire que sous certaines conditions en rajoutant des structures conditionnelles.

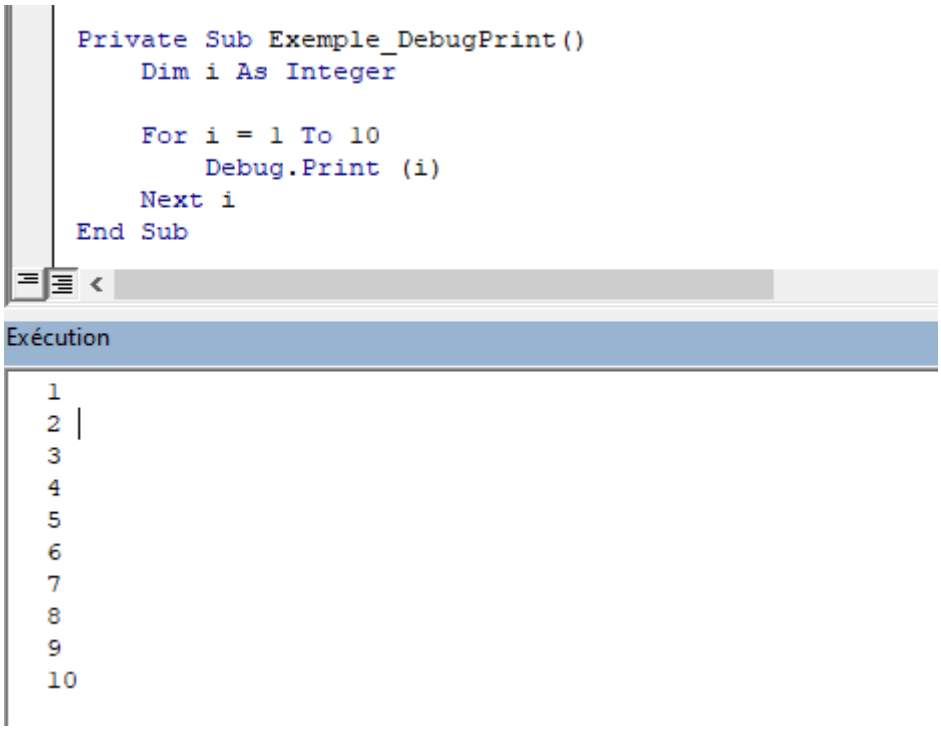

FIGURE 2.7. – Fenêtre exécution

Pour nettoyer le contenu de la fenêtre d'exécution, nous pouvons cliquer dedans, tout sélectionner avec Ctrl+A et supprimer avec <-.

2.1.3. Ailleurs

Enfin, nous pouvons aussi imaginer écrire des données ailleurs (feuille de calcul, fichier texte, base de données, etc.). Cela est tout de suite plus complexe et doit être le fait de besoins bien spécifiques.

Pensez à supprimer vos instructions `Debug.Print` ou `MsgBox` dès que vous n'en avez plus besoin pour ne pas gêner l'exécution du programme ou afficher des données par inadvertance.

2.2. ...en les visualisant

Il peut être fastidieux de tracer les données soi-même quand il faut procéder à une analyse poussée. En mode débogage, il existe une fenêtre accessible via "Fenêtre Variables locales" du menu "Affichage".

Celle-ci liste pour nous à un instant T les valeurs pour la procédure ou fonction dans laquelle nous nous trouvons.

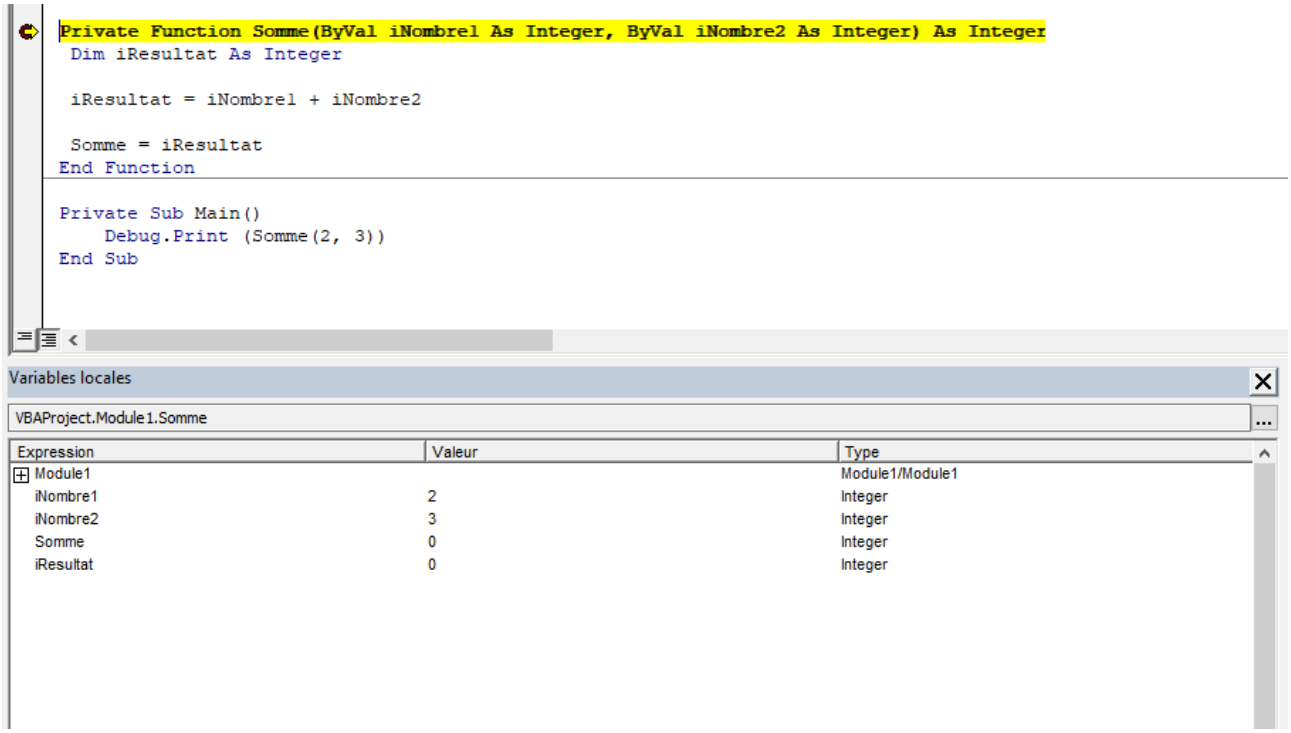

FIGURE 2.8. – Fenêtre variables locales

? Qu'est-ce que ce point rouge et cette ligne jaune ?

Pour les besoins de cette section, j'ai un peu triché et j'ai utilisé le débogueur avant l'heure.

Le point rouge est ce que l'on appelle un point d'arrêt. Il sert à indiquer qu'il faut faire une pause dans l'exécution à cet endroit là. Je l'ai placé sur l'entête de la fonction *Somme*, mais j'aurais pu le mettre plus bas si souhaité. Lorsque la procédure *Main* est exécutée, la fonction est appelée et le point d'arrêt provoque le passage en mode pas à pas (la ligne jaune).

Les valeurs affichées correspondent à leur état avant exécution de la ligne jaune, comme illustré ci-dessous :

2. *Analyser l'évolution des données...*

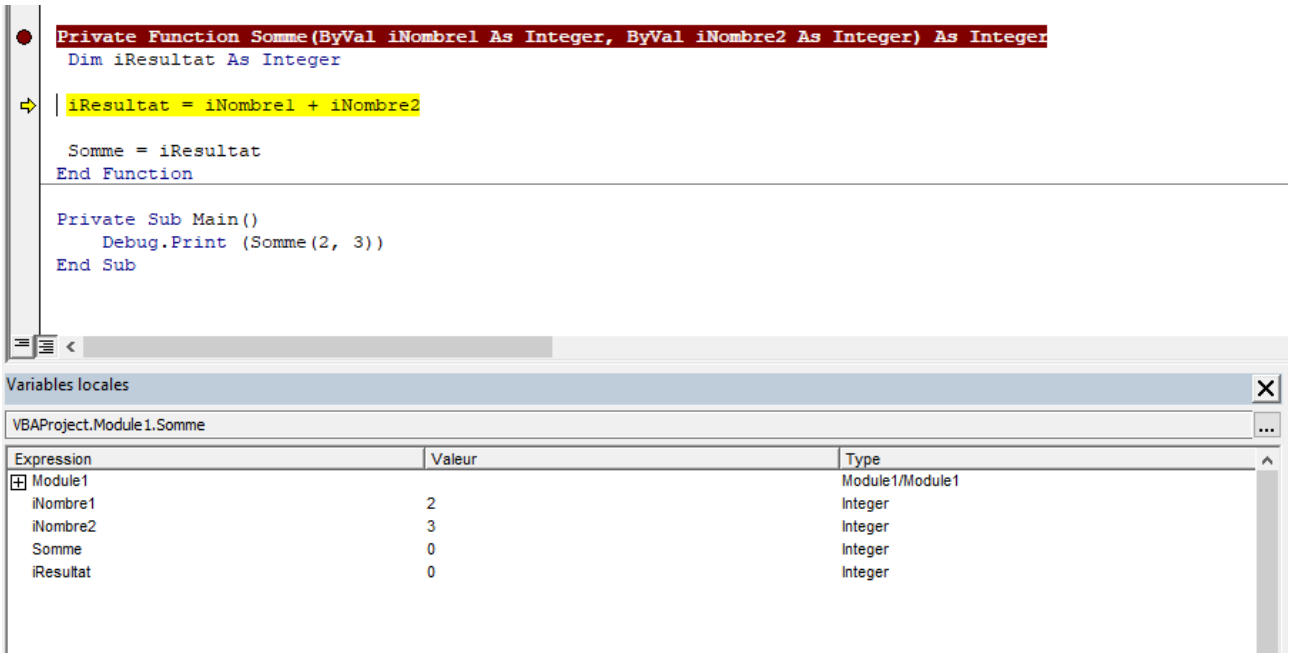

FIGURE 2.9. – Fenêtre variables locales (2)

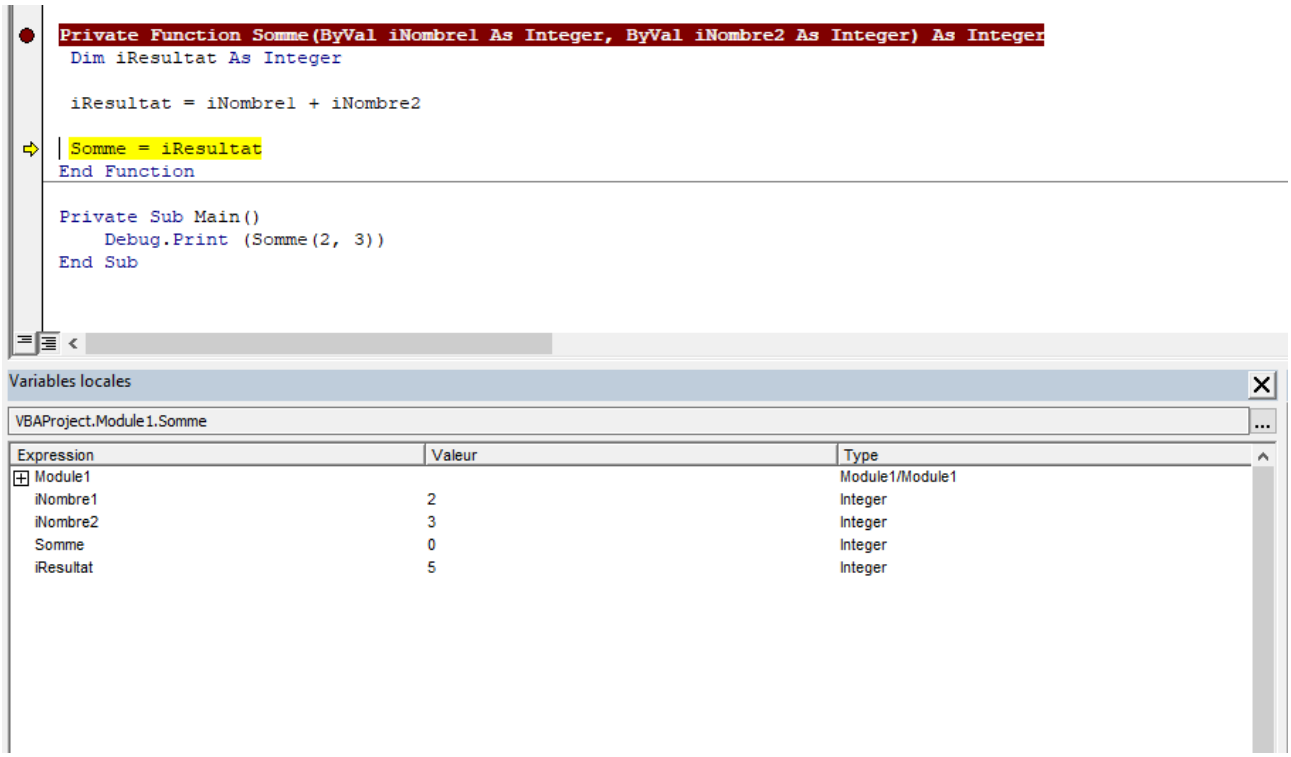

FIGURE 2.10. – Fenêtre variables locales (3)

2.3. ...en les espionnant

Une dernière possibilité est d'espionner les valeurs.

Contrairement à la fenêtre des variables locales, c'est à nous d'indiquer les variables à suivre. Néanmoins, nous avons accès ici à une portée plus étendue ainsi qu'à des possibilités plus intéressantes.

Pour ajouter un espion sur une variable, nous pouvons soit faire clique droit dessus et choisir "Ajouter un espion", soit passer par le sous-menu "Ajouter un espion" du menu "Débogage". Avec le clique droit, le contexte est automatiquement rempli.

2.3.1. Espionnage simple

Pour ce premier exemple, nous faisons de l'espionnage simple en regardant une variable.

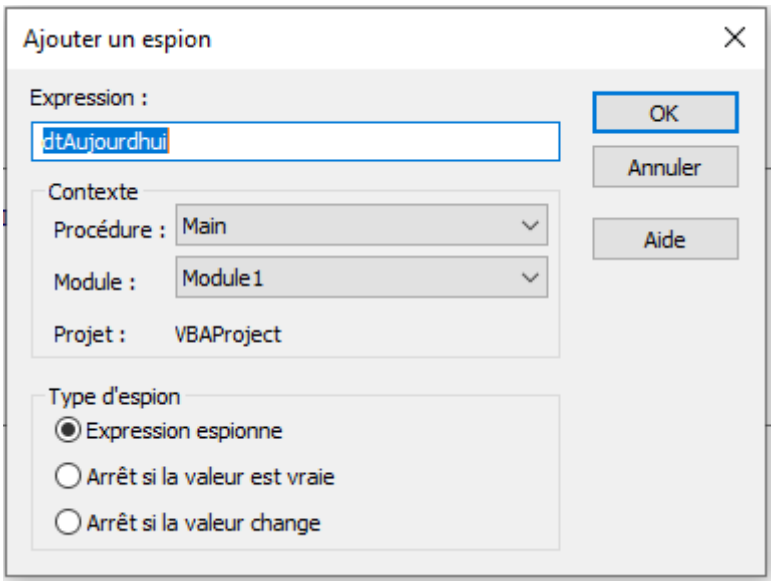

FIGURE 2.11. – Ajouter un espion simple

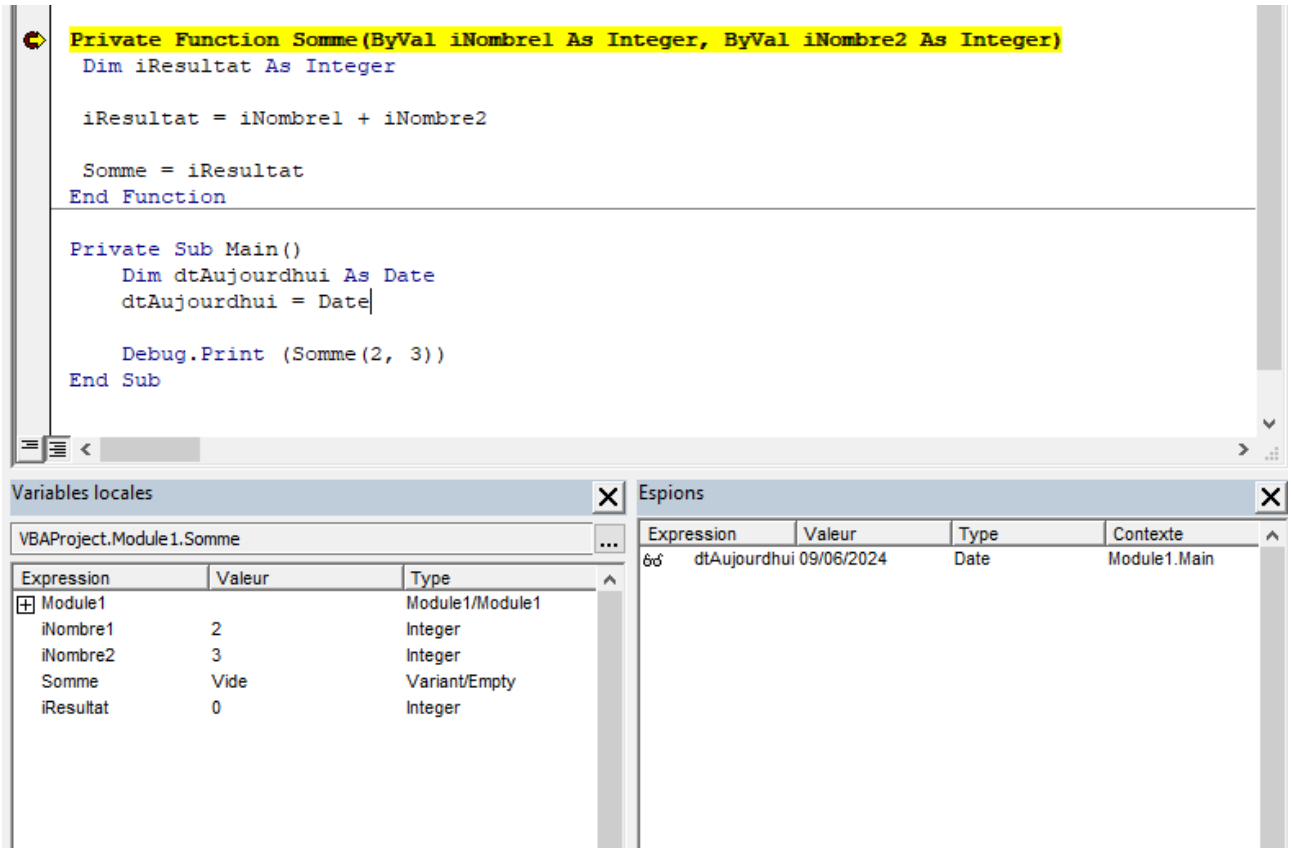

FIGURE 2.12. – Exemple espionnage simple

Nous pouvons voir que nous avons bien accès à des données dans une portée plus large que la fenêtre des variables locales.

2.3.2. Espionnage conditionné

Autre possibilité avec l'espionnage, c'est de provoquer un arrêt sous certaines conditions, soit si la valeur de l'expression est vraie, soit si la valeur de l'expression change. Cela convient pour suivre l'évolution d'une variable ou s'arrêter à une itération voulue d'une boucle par exemple.

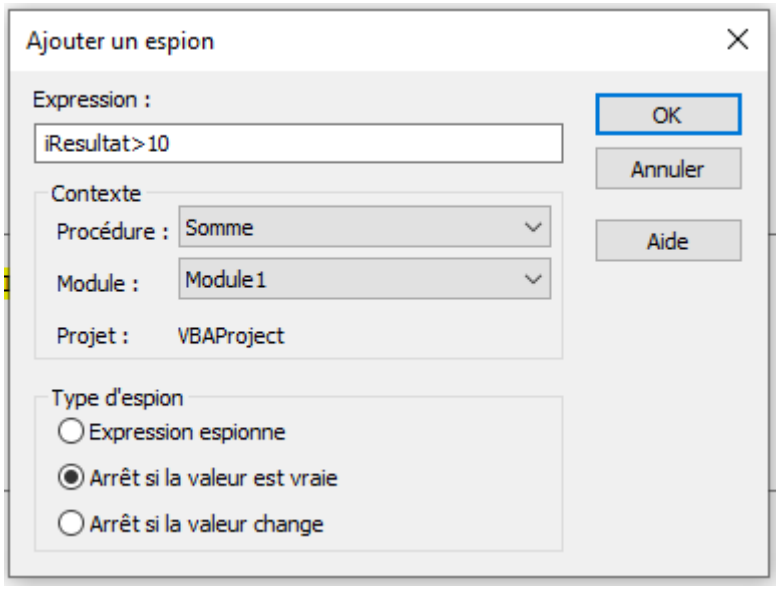

FIGURE 2.13. – Ajouter un espion conditionné

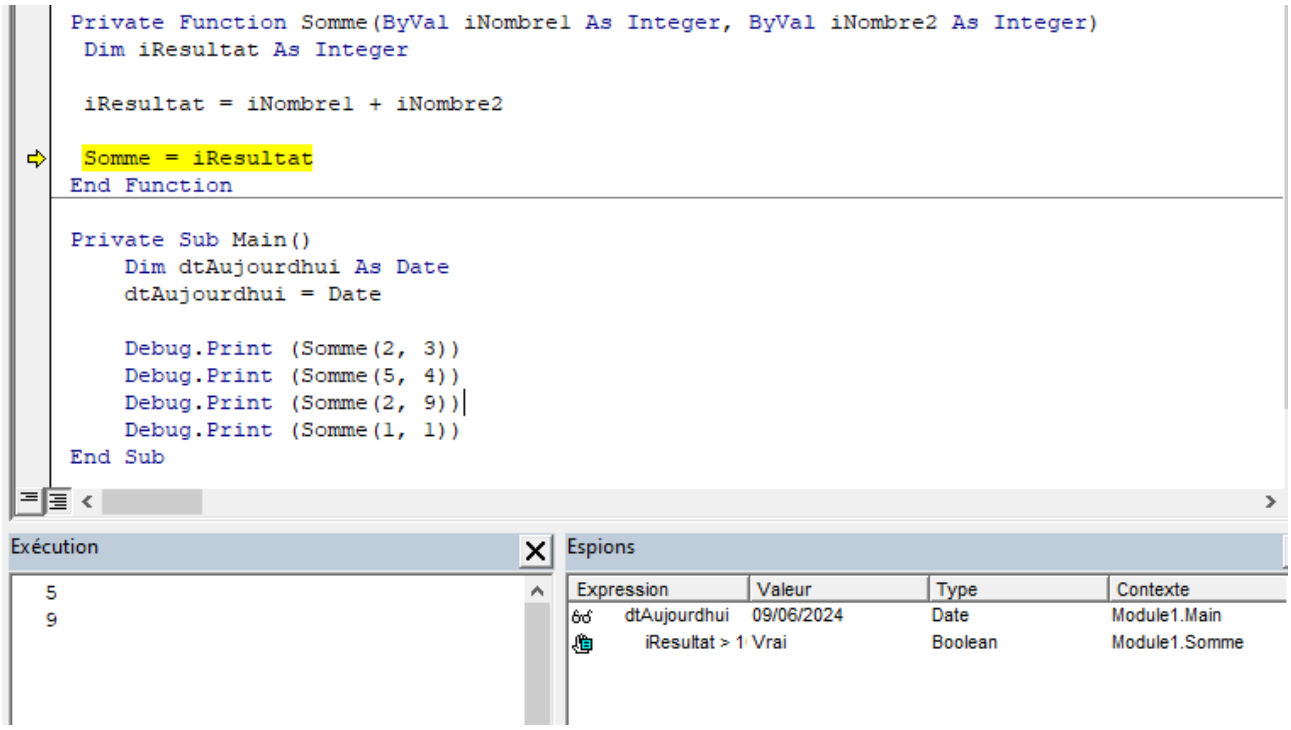

FIGURE 2.14. – Exemple espionnage conditionné

Dans cet exemple, j'ai enlevé le point d'arrêt manuel et nous pouvons constater que le résultat de la somme dépassant 10 provoque bien le passage en mode pas à pas.

Il y a donc plusieurs façons de suivre l'évolution des données. Certaines sont faites pour être utilisées avec le débogueur. L'usage dépendra des besoins.

3. Recourir au débogueur

Le débogueur permet de maîtriser l'exécution du code (en plaçant des arrêts à des endroits, en exécutant une ligne après l'autre, etc.).

Il est vrai que nom peut faire un peu peur, mais c'est grâce à lui que nous allons pouvoir voir le résultat d'opérations et l'évolution des données à des endroits précis.

3.1. Début débogage

Le débogage peut se lancer suite à une erreur d'exécution ou de façon manuelle.

3.1.1. Débogage suite erreur exécution

Lorsque l'exécution du programme est lancée et qu'une erreur se produit, une fenêtre apparaît.

```
1 Private Sub Exemple_ErreurExecution()
2     Dim iNombre1 As Integer
3     Dim iNombre2 As Integer
4     Dim iResultat As Integer
5     iNombre1 = 10
6     iNombre2 = 0
7     iResultat = iNombre1 / iNombre2
8 End Sub
```

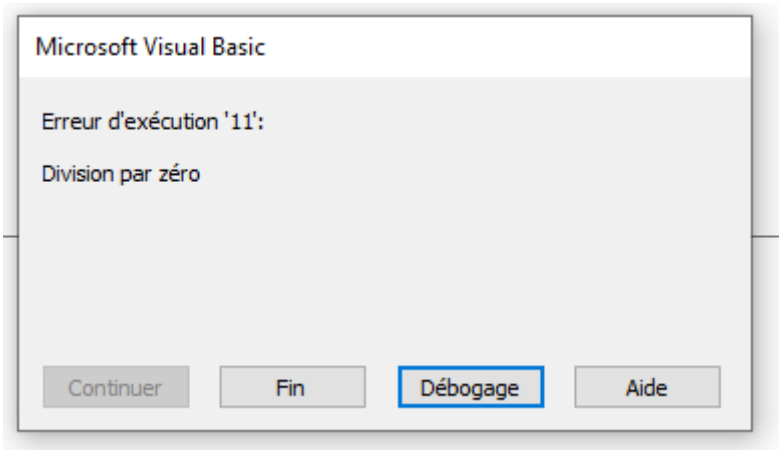

FIGURE 3.15. – Erreur exécution

Dans celle-ci, nous pouvons notamment trouver un message plus ou moins parlant ainsi qu'un bouton "Aide".

De plus, nous pouvons aussi voir un bouton "Débogage" et c'est lui qui nous intéresse tout particulièrement. En cliquant dessus, le programme se place sur la ligne ayant planté. Nous pouvons alors analyser les données via les différents moyens étudiés précédemment ainsi qu'en déplaçant la souris sur les variables.

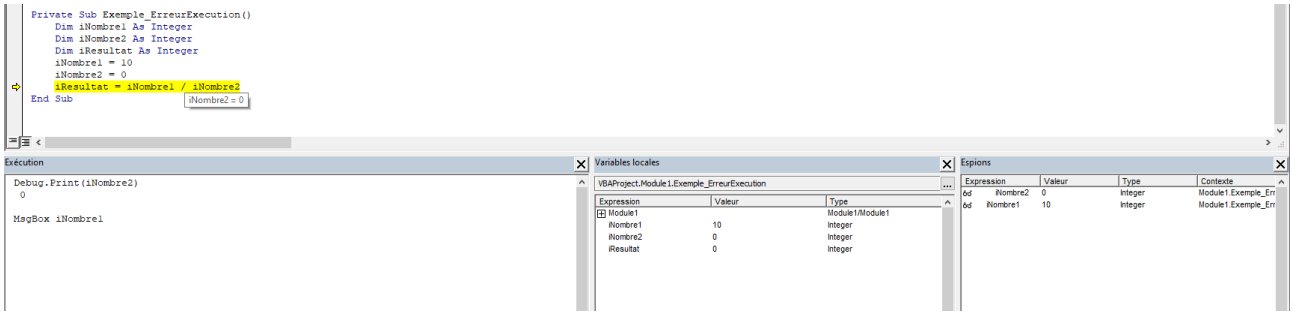

FIGURE 3.16. – Analyser les données en mode débogage

J'ai été assez généreux pour vous montrer différentes possibilités, remarquez l'utilisation de code dans la fenêtre d'exécution 🍊

Pour mettre fin à l'exécution et au débogage, il faut cliquer sur le bouton "Réinitialiser".

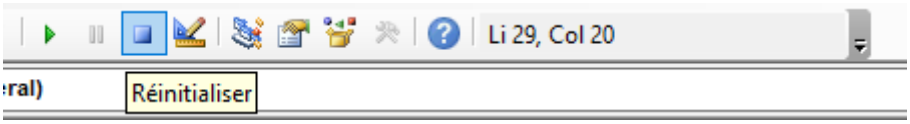

FIGURE 3.17. – Fin du débogage

3.1.2. Débogage manuel

Pour lancer le débogage manuel, nous avons parlé des points d'arrêt qui sont à placer et des espions qui permettent de lancer le débogage sous conditions.

3.1.2.1. Point d'arrêt Les points d'arrêt sont ces petits points rouges dans la marge grise. Ils sont là pour signaler qu'il faut faire une pause à ces endroits.

Ils peuvent être ajoutés et supprimés en cliquant dans cette marge grise à côté d'une ligne ou encore via les sous-menus "Basculer le point d'arrêt") (raccourci F9) pour une ligne après avoir cliquée sur celle-ci. Certaines lignes ne permettent pas de placer un point d'arrêt (déclaration de variable, ligne vide, …).

Il est possible de supprimer tous les points d'arrêt via le raccourci Ctrl+Maj+F9.

3.2. Usage

Une fois le programme figé, il y a plusieurs possibilités.

```
Private Function Somme(ByVal iNombre1 As Integer, ByVal iNombre2 As Integer)
 Dim iResultat As Integer

 iResultat = iNombre1 + iNombre2

 Somme = iResultat
End Function

Private Sub Main()
    Dim dtAujourdhui As Date
    dtAujourdhui = Date

    Debug.Print (Somme(2, 3))
    Debug.Print (Somme(5, 4))
    Debug.Print (Somme(2, 9))
    Debug.Print (Somme(1, 1))
End Sub
```

FIGURE 3.18. – Point d'arrêt placé sur Main

3.2.1. Continuer ou arrêter

Outre continuer l'exécution jusqu'au prochain point d'arrêt ou jusqu'à la fin, nous avons vu qu'il était possible de stopper l'exécution :

FIGURE 3.19. – Boutons continuer ou arrêter exécution

Ensuite, le menu débogage permet d'avancer au rythme voulu :

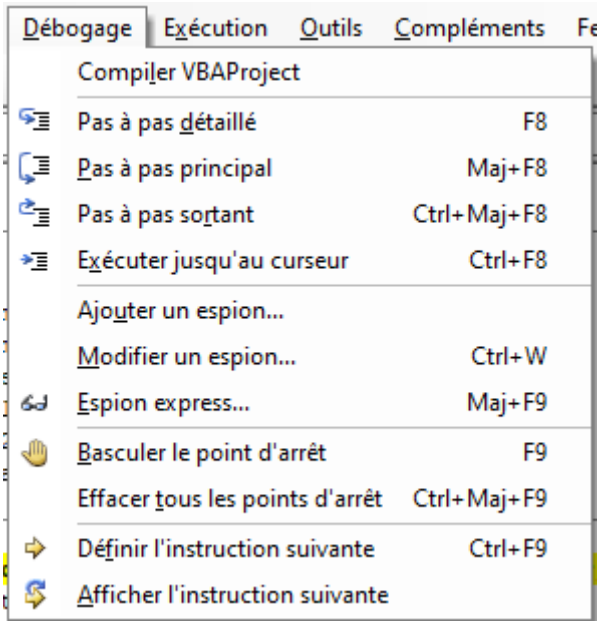

FIGURE 3.20. – Menu débogage

3.2.2. Pas à pas détaillé

Ce mode permet de dérouler au détail ligne par ligne en rentrant dans les procédures ou fonctions appelées également. Dans notre exemple, cela a pour effet de dérouler y compris dans la fonction *Somme*.

3.2.3. Pas à pas principal

Ce mode a pour effet de dérouler sans aller dans le détail des procédures ou fonctions appelées. Dans notre exemple, seules les lignes de la procédure *Main* seront parcourues.

3.2.4. Pas à pas sortant

Ce mode a pour effet de sortir de la fonction ou procédure en cours (ou d'aller jusqu'au prochain arrêt) pour se placer sur l'instruction suivante. Dans notre exemple, si la fonction *Somme* avait un point d'arrêt, et que nous faisions pas à pas sortant, nous nous retrouverions sur la ligne suivant l'appel, donc `Debug.Print (Somme(5, 4))`.

3.2.5. Exécuter jusqu'au curseur

Cette option permet d'aller jusqu'au prochain point d'arrêt ou à défaut à la fin du programme.

3.2.6. Définir l'instruction suivante

Cette option permet, après avoir au préalable cliqué sur une ligne, de dérouler le programme jusqu'à cette dernière. Elle devient la ligne jaune. Remarquons que les points d'arrêt intermédiaires sont ignorés.

À travers cette dernière section, nous avons étudié l'usage du débogueur de l'éditeur Visual Basic Editor.

Conclusion

C'est déjà la fin de ce billet.

Au cours de celui-ci, nous avons vu comment nous prémunir d'erreurs liées à la syntaxe et de logique ainsi que comment comprendre les erreurs provenant de la logique de notre code, en analysant les données et les instructions.

Certaines erreurs durant l'exécution demande parfois une gestion particulière. Par exemple, si l'utilisateur saisit une mauvaise donnée, nous pourrions vouloir l'alerter plutôt que le programme s'arrête des suites d'un bogue. Le langage VBA offre alors des instructions pour faire cela.

À bientôt !

Quelques ressources :

— Livre Programmation VBA pour Excel pour les nuls (Excel 2010, 2013 et 2016), de John Walkenbach
— La documentation

Liste des abréviations

VBA Visual Basic for Applications. 1

VBE Visual Basic Editor. 1