LES OUTILS DE DEBOGAGE VBA

Quelle que soit notre expérience de **programmation dans le langage VBA** (Visual Basic for Application), **nous commettons inévitablement plus ou moins d'erreurs** dans l'écriture de nos codes.

EXCEL offre un **ensemble d'outils indispensables** pour trouver ces erreurs et les corriger.

Cette phase indispensable de mise au point est désignée par le terme **« débogage »**.

Cette fiche « **Bonnes pratiques VBA** » présente de façon concise ce qu'il est nécessaire de connaître sur ces outils et sur leur bonne utilisation en programmation VBA.

(Les copies-écrans du présent document sont issues d'EXCEL version 365)

L'icône signale une remarque réclamant une **attention particulière**.

L'icône marque les **commentaires en marge** du sujet traité. Ils peuvent être ignorés sans inconvénient.

Table des matières

DEBUTONS LE DEBOGAGE : COMPILONS ! 2
Pourquoi compiler ? 2
Quand compiler ? 3
Comment compiler ? 3
DEBOGUONS LES ERREURS D'EXECUTION ET DE LOGIQUE 4
Ajoutons la barre de menu du débogueur 4
Exécutons le code en mode pas à pas 5
Examinons les techniques et outils à notre disposition 7
Affichage de la valeur d'une variable 7
Mise en place de point d'arrêt 7
Mise en place d'espions 8
La fenêtre des « Variables locales » : 9
La fenêtre d'exécution 10
Utilisation de l'instruction « Debug.Print » 12
Utilisation de l'instruction « Msgbox » 13
APRES DEBOGAGE, N'OMETTONS PAS ! 13

DEBUTONS LE DEBOGAGE : COMPILONS !

Le langage VBA est du **type « interprété »** alors que d'autre langages sont de **type « compilé »** (tels C++, C#, Pascal…)

Cela signifie qu'**il n'y a pas nécessité** de compiler les codes VBA **pour pouvoir les exécuter**.

L'interpréteur VBA se charge de **transformer « à la volée »**, **ligne à ligne**, le code VBA **compréhensible par l'humain** en code **compréhensible par la machine** (exécutable).

Toutefois, VBA possède une **fonction de compilation** produisant à partir du code source VBA un autre objet exécutable par Windows.

Il semble que cette opération **n'ait pas d'impact significatif** sur les **performances d'exécution** des codes VBA, contrairement à ce qu'on pourrait attendre.

Pourquoi compiler ?

En compilant notre code, nous pourrons **éliminer les erreurs de syntaxe** non corrigées ou non détectées lors de la saisie du code dans l'éditeur VBE.

Ici, une **liste non exhaustive des erreurs** commises couramment **détectées lors de la compilation** :

- Instruction IF sans END IF
- Directive WITH sans END WITH
- Instruction SELECT sans END SELECT
- Instruction FOR ou FOR EACH sans NEXT
- Appel de procédure ou fonction inexistantes ou non atteignables
- Passage de paramètres non conformes ou omis lors d'appel aux procédures et fonctions.
- Variable non déclarée en cas d'utilisation de la clause « Option Explicit ».
- …

VBA ne nous **contraint pas** à déclarer toutes les variables utilisées dans le code (d'ailleurs, par exemple, l'enregistreur de macro ne le fait pas).

Cependant, beaucoup d'auteurs de référence recommandent l'utilisation de la **clause « Option explicit »**

Concernant **l'utilisation des variables** en VBA, nous nous reporterons utilement aux **chapitres de la formation VBA du site « EXCEL-PRATIQUE » :**

https://www.excel-pratique.com/fr/vba/variables

https://www.excel-pratique.com/fr/vba/variables_suite

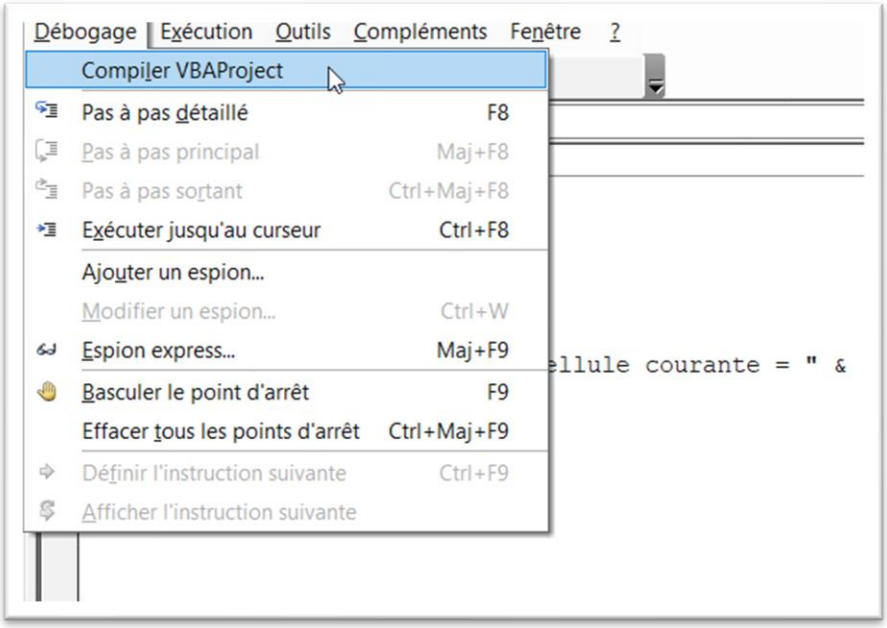
Quand compiler ?

Le compilateur VBA n'indique pas les erreurs en bloc mais **s'arrête à chaque fois qu'il en rencontre une** pour la signaler.

Nous avons tout intérêt, pour corriger nos erreurs de syntaxe, à **lancer une compilation** régulièrement et dès que nous saisissons une **séquence de code un peu conséquente**.

Comment compiler ?

Nous exécutons la compilation de notre projet VBA par l'activation des choix dans le menu VBE : **Débogage/Compiler [nom de projet]** ou le raccourci clavier : **'Alt+D+l'**



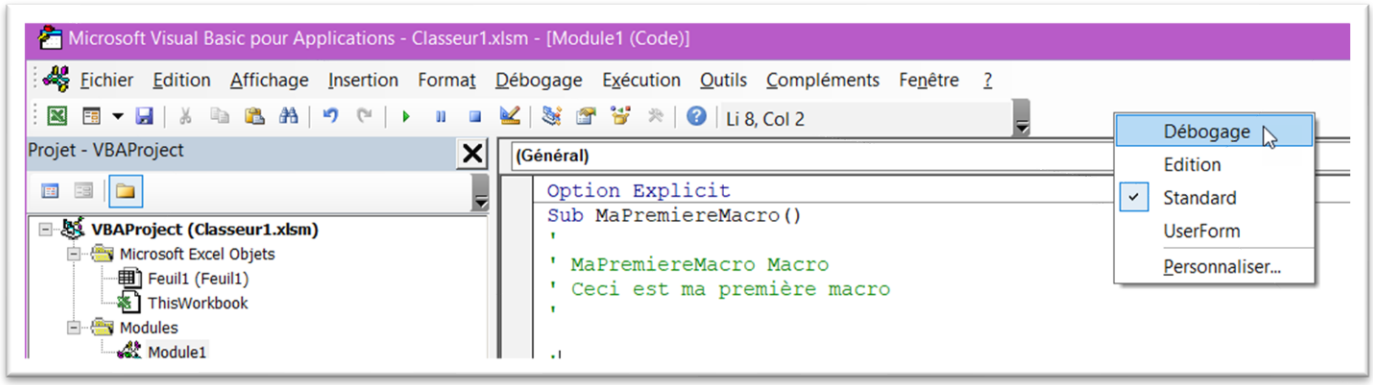
Lorsqu'un **projet est compilé sans erreur**, le choix « Compiler [nom de projet] » **apparaît en grisé** dans le sous-menu « Débogage ».

DEBOGUONS LES ERREURS D'EXECUTION ET DE LOGIQUE

Une fois notre code compilé sans erreur, nous nous attaquons au débogage des erreurs se produisant à l'**exécution** ainsi que des **erreurs de logique**.

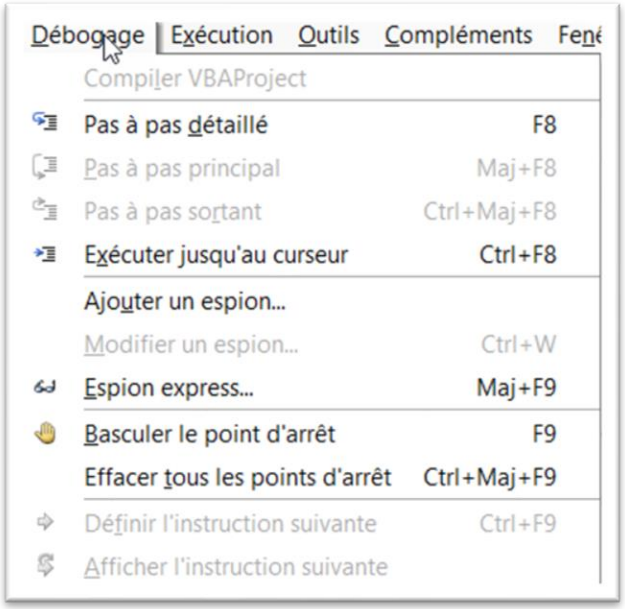
Ajoutons la barre de menu du débogueur

Pour nous faciliter la tâche, nous pouvons ajouter à l'éditeur VBA **la barre de menu spécifique au débogage** : En cliquant droit sur la barre de menu et en sélectionnant **« Débogage »** :



La barre suivante est ajoutée :

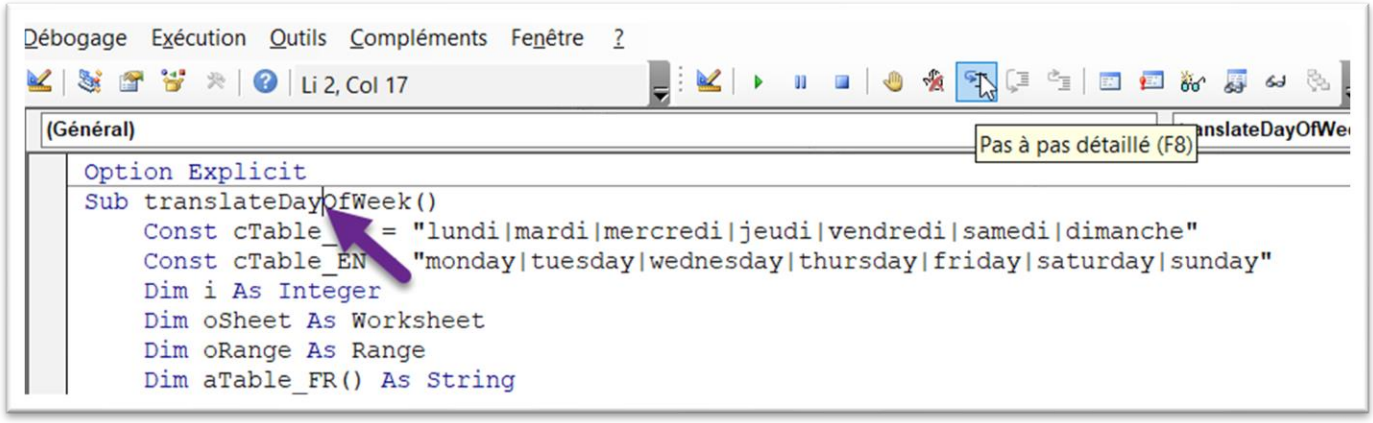
Cette barre reprend les fonctions accessibles dans le sous-menu « Débogage » :



Exécutons le code en mode pas à pas

Dans l'éditeur VBE, nous avons la possibilité **d'exécuter le code suivant diverses modalités** :

En **positionnant le curseur sur le début** d'une procédure ou fonction et en cliquant sur le bouton (ou F8) nous exécutons le code en **mode pas à pas**, sur chaque ligne d'instructions de notre code :

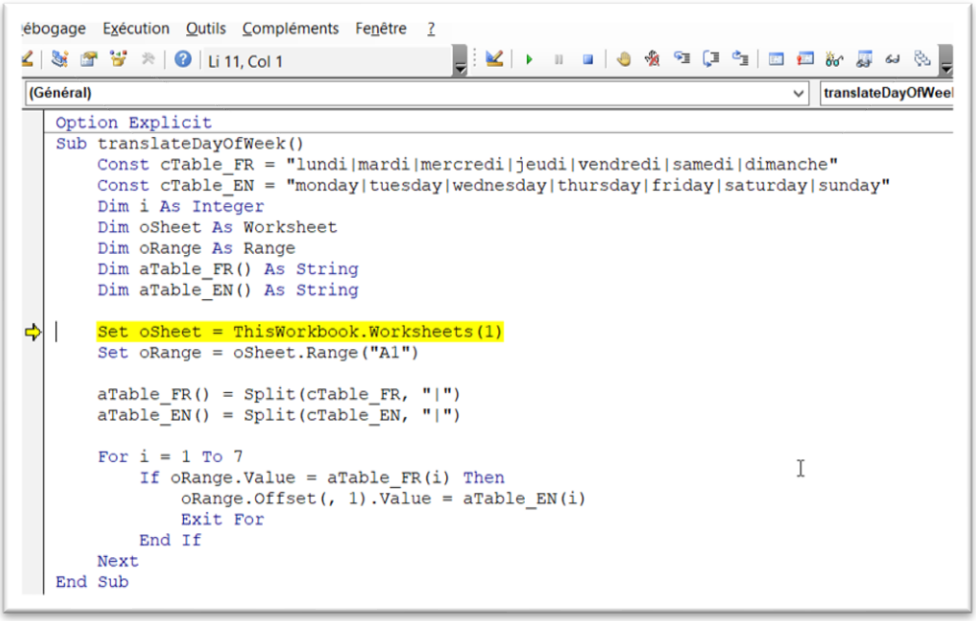


De cette façon, nous pourrons **suivre le cheminement de notre code**, en **vérifier la logique** et **corriger les erreurs**.

Nous ne pouvons exécuter de cette façon que les procédures (Sub) ou fonctions (Function) qui n'attendent **aucun paramètre d'entrée**. Aussi, pour déboguer ce type de procédure ou fonction, **nous débuterons** le débogage **dans une procédure ou fonction appelante** transmettant les paramètres attendus.

Après avoir activé le mode pas à pas, la première ligne de la procédure ou fonction **à exécuter** est **surlignée en jaune**.

Lorsque nous cliquons une nouvelle fois sur le bouton « Pas à pas » (ou F8), le débogueur **se positionne sur la première instruction** à exécuter**, en négligeant le lignes de déclarations** de constantes ou de variables :



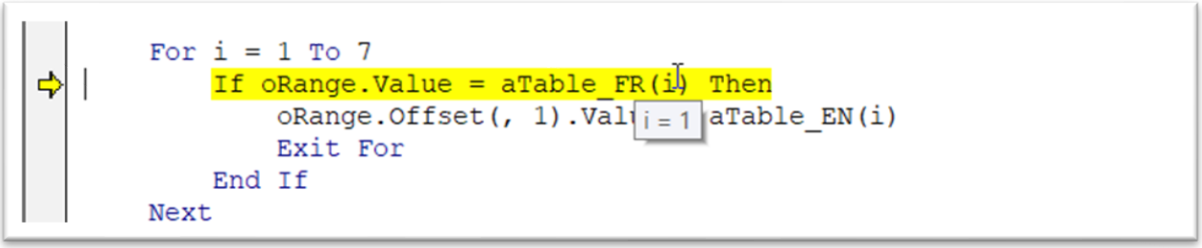
Ainsi de suite jusqu'à la fin de la procédure ou fonction (End Sub/End Function)

En mode d'exécution pas à pas, en **plaçant le curseur** sur la **flèche jaune** dans la marge tout en maintenant le **clic gauche de la souris**, nous avons la possibilité de **déplacer la prochaine exécution** en **aval** ou en **amont** des instructions du code.

Examinons les techniques et outils à notre disposition

Affichage de la valeur d'une variable

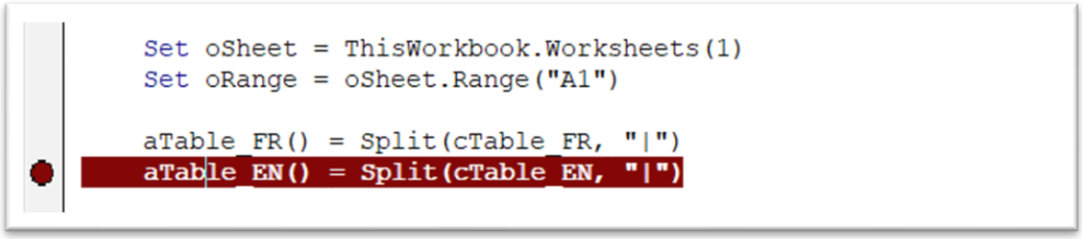
En suivant l'exécution de notre code en mode « pas à pas », nous avons la possibilité de connaître la valeur de variables impliquée dans l'instruction à exécuter, en **plaçant simplement le curseur** sur cette variable : **un « tip »** nous indique sa valeur :



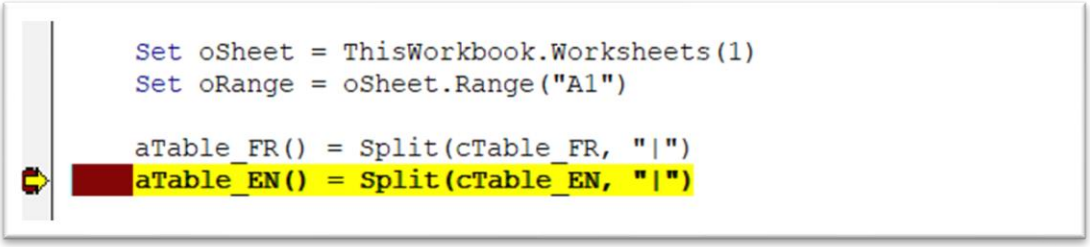
Mise en place de point d'arrêt

Nous positionnons un point d'arrêt en cliquant **dans la marge** devant une instruction (ou en cliquant sur le bouton ou F9).

La ligne d'instruction apparaît alors **surlignée en rouge foncé** avec un repère de même couleur dans la marge :



Une fois un point d'arrêt positionné, lorsque nous exécutons le code en mode « continu » (F5), l'exécution **s'arrêtera** sur la ligne avant de l'exécuter :



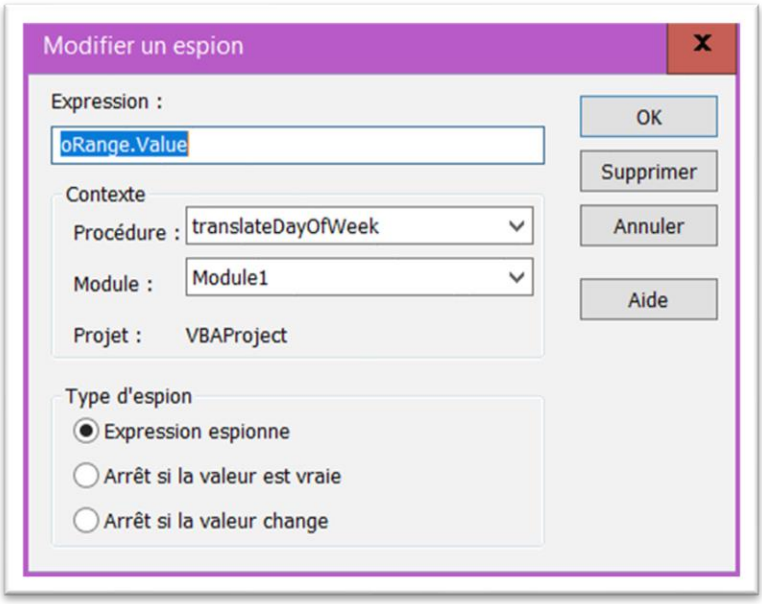
Pour effacer tous les points d'arrêts positionnés : (Ctrl+Maj+F9).

Mise en place d'espions

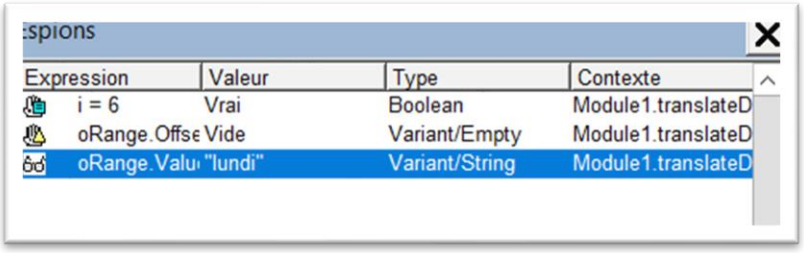
Avec la mise en place d'espions, nous avons la possibilité de définir des expressions permettant soit de **visualiser leurs valeurs** tout au long de l'exécution, soit de **stopper l'exécution** lorsque l'expression devient **vraie**, soit de **stopper l'exécution** lorsque leurs valeurs **changent**.

Nous positionnons un espion en cliquant le bouton (ou Alt+D+u).

Dans l'exemple suivant nous positionnons un espion permettant de suivre **l'évolution de la valeur** de la variable objet « oRange » :

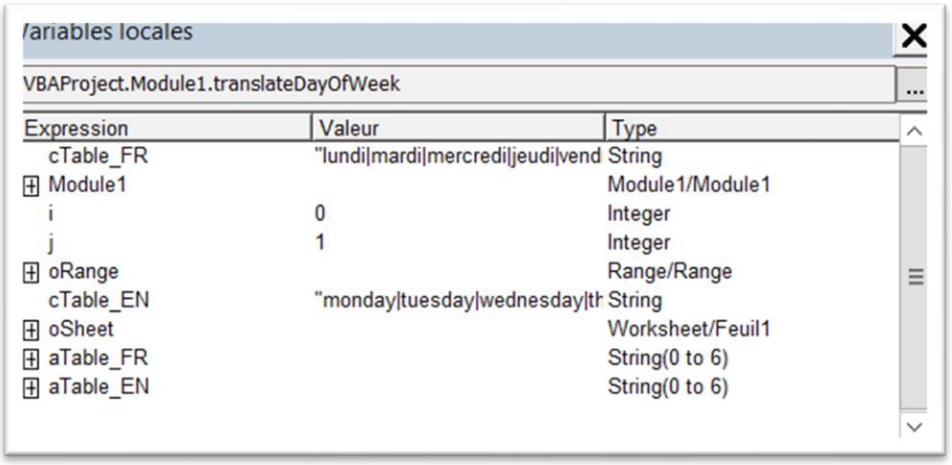


La fenêtre « Espions » que nous faisons apparaître en cliquant sur le bouton (ou Alt+A+s), liste tous les espions que nous avons positionnés ainsi que leurs valeurs :

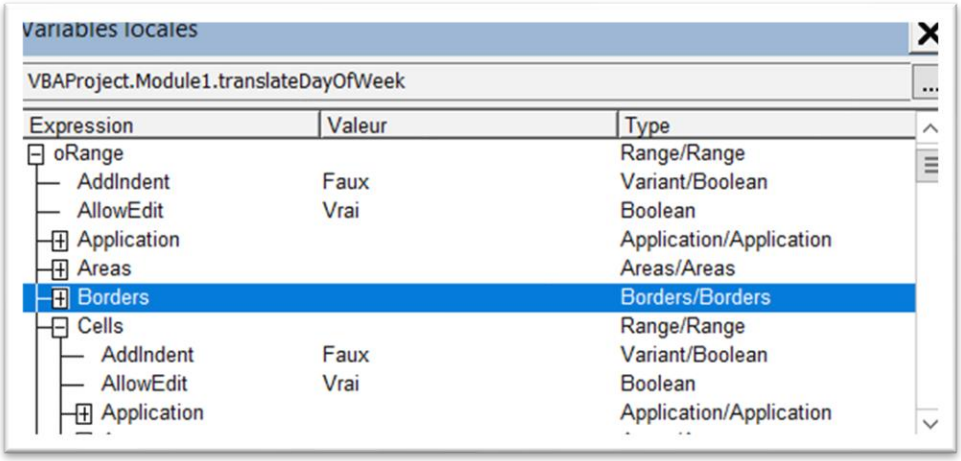


La fenêtre des « Variables locales » :

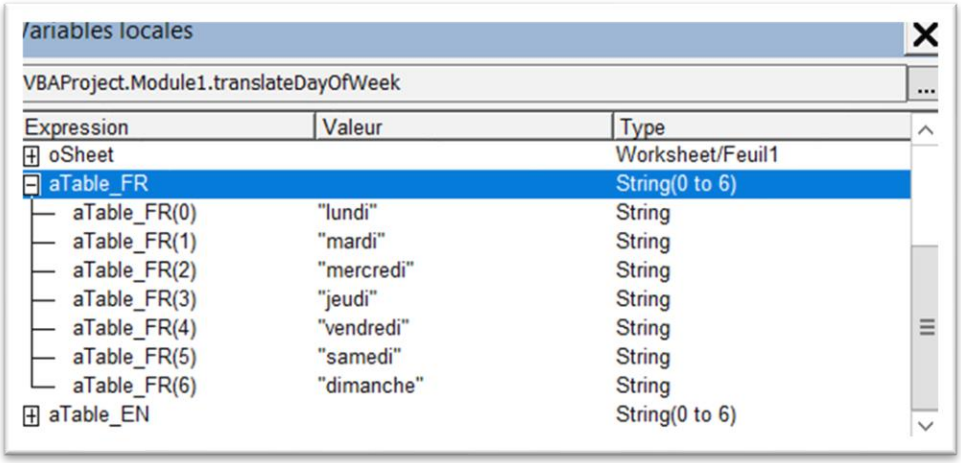
La fenêtre « Variables locales » que nous faisons apparaître en cliquant sur le bouton (ou Alt+A+v), liste toutes les variables de la procédure en cours :



De plus, nous avons la possibilité dans cette fenêtre d'ouvrir, pour les **variables objets**, la liste de **toutes les propriétés** de l'objet avec **leurs valeurs** :



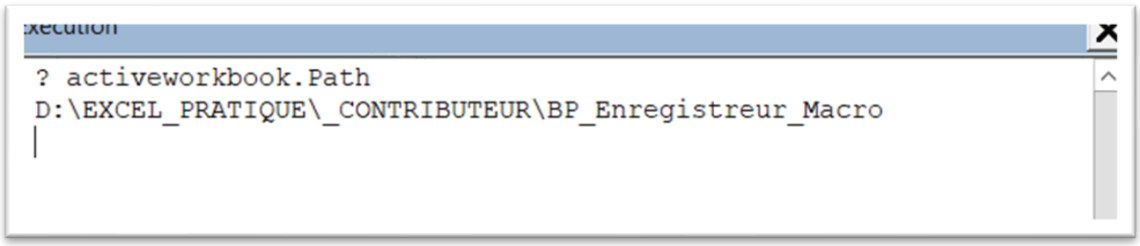
Et pour les **tableaux mémoire** (array), la liste de **toutes leurs dimensions** et **valeurs** :



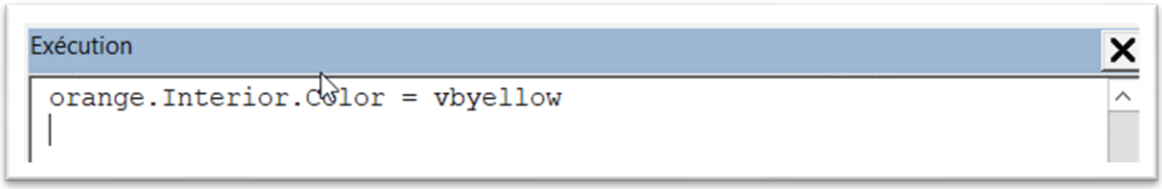
La fenêtre d'exécution

La fenêtre « d'exécution » que nous faisons apparaître en cliquant sur le bouton (ou Ctrl+G), nous offre une multitude d'utilisations :

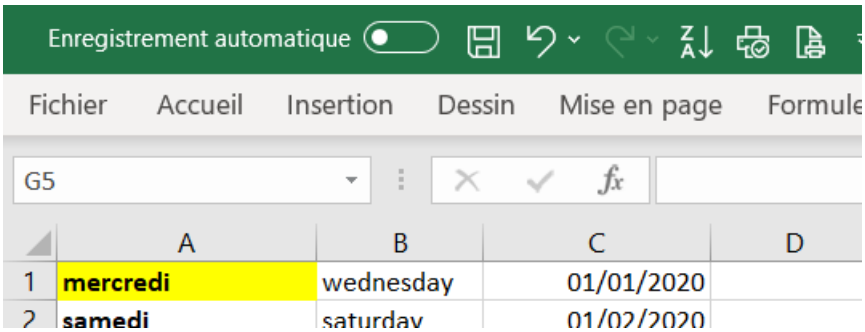
- En y **formulant une question**, nous pouvons obtenir **immédiatement la réponse**.
  Par exemple si nous voulons connaitre le dossier du classeur actif, il suffit d'enter dans la fenêtre le signe « ? » suivi de la propriété recherchée. La valeur est alors restituée dans la ligne suivante de la fenêtre :



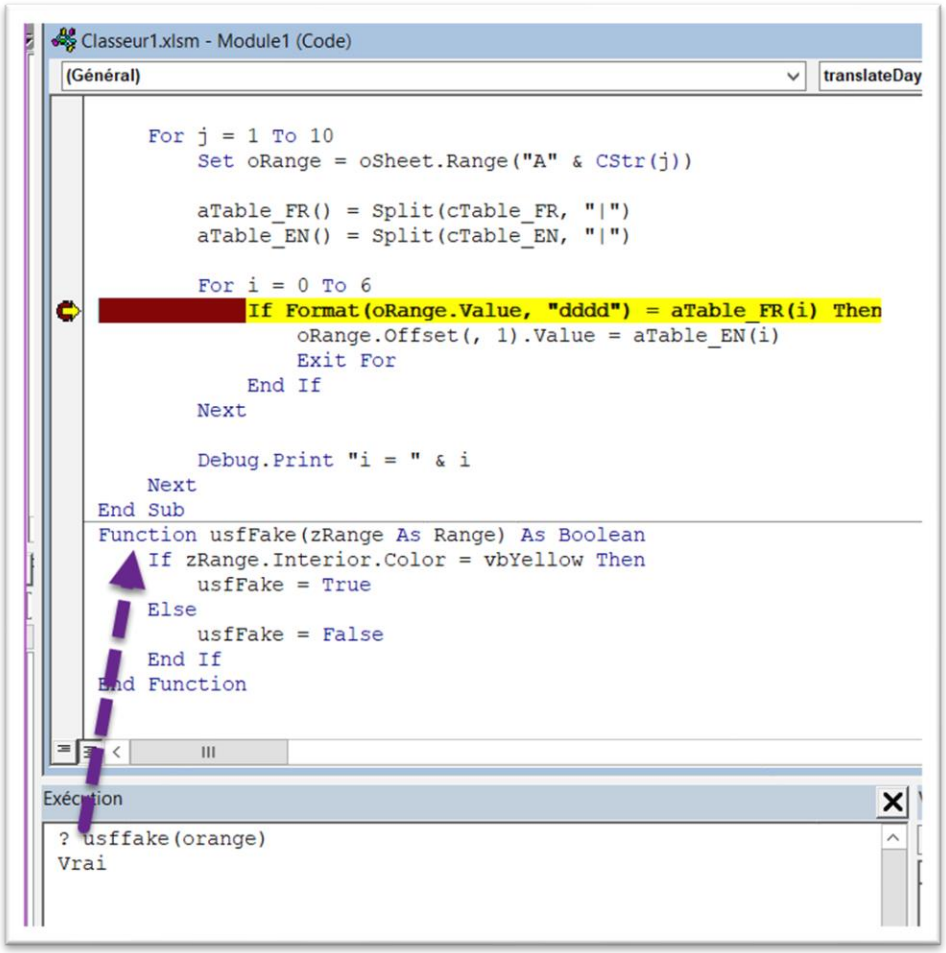
- Nous pouvons également y **exécuter une instruction** en dehors du code de la procédure exécutée :



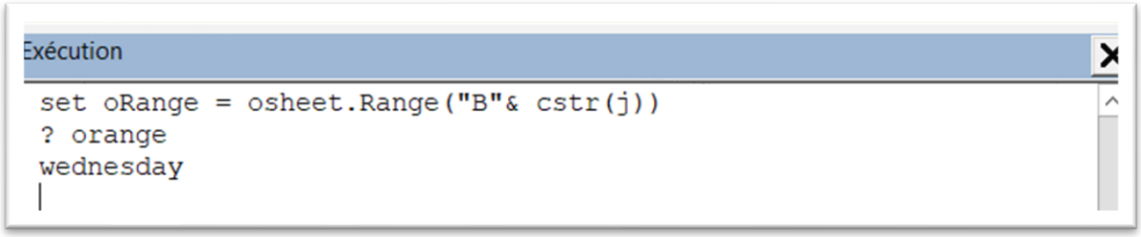
Résultat :



- Nous pouvons **exécuter une procédure ou une fonction** en lui passant des paramètres et obtenir la valeur retournée :



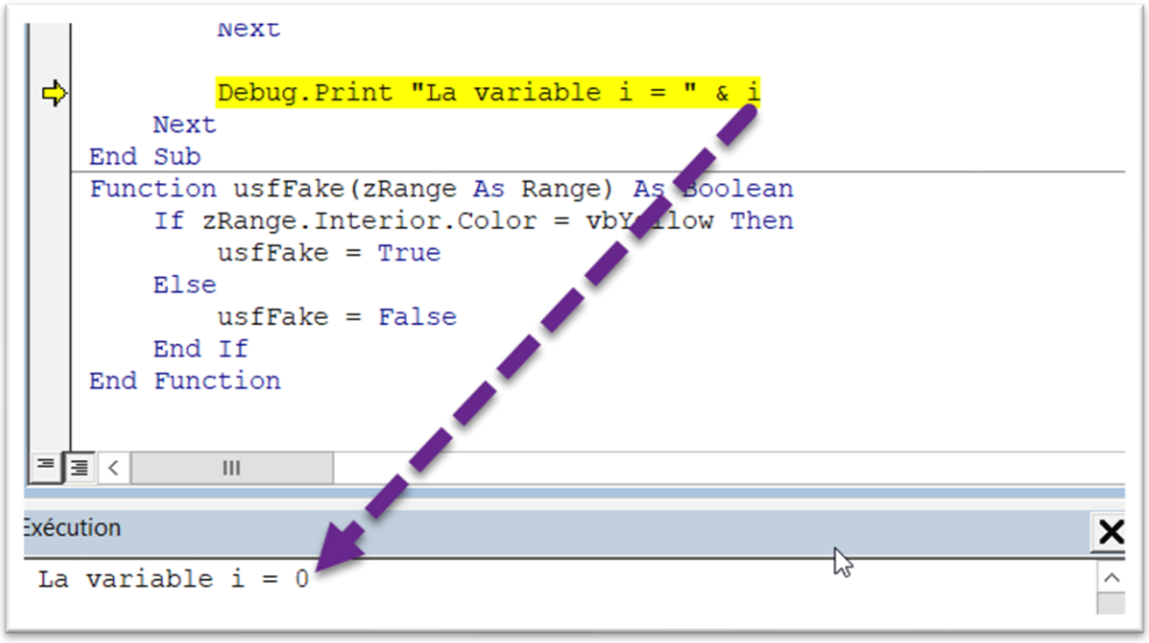
- Nous pouvons **affecter une valeur à une variable :**



- Nous pouvons **y voir figurer les valeurs** affectées par l'instruction « **Debug.Print** » (Cf ci-dessous).

Utilisation de l'instruction « Debug.Print »

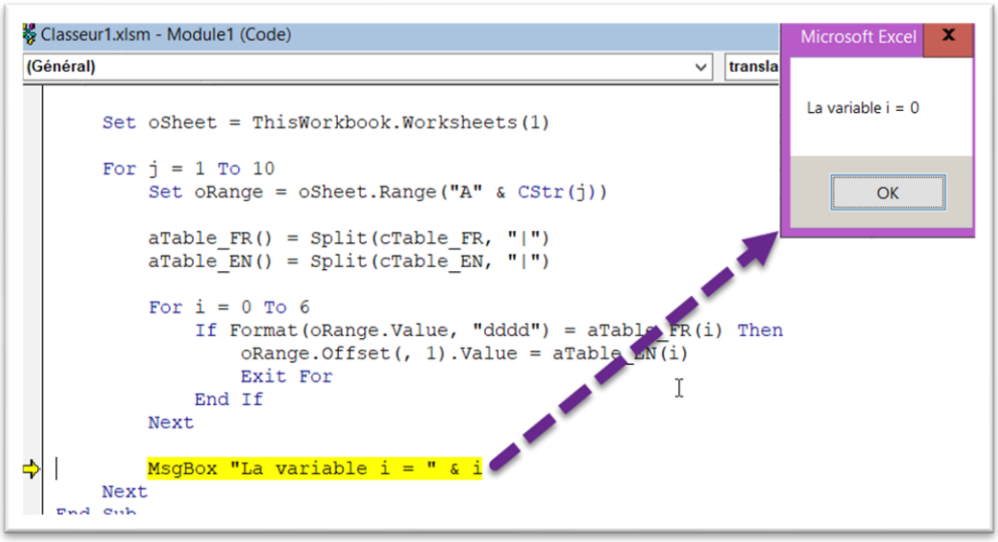
L'instruction « **Debug.Print** » suivi d'une valeur introduite dans le code, nous permet de **visualiser cette valeur** dans la « **Fenêtre exécution** » (Ctrl+G)



Nous pouvons noter que l'avantage de cette instruction réside dans le fait qu'**elle peut être laissé dans le code** sans inconvénient du fait que **le résultat n'apparaîtra pas** lors d'un **usage ordinaire**.

Utilisation de l'instruction « Msgbox »

L'instruction « **Msgbox** » suivi d'une valeur, nous permet d'obtenir l'affichage d'un message contenant cette valeur :



Le **déroulement du code est interrompu** tant que l'on ne clique pas sur le bouton « OK » du message.

Nous pouvons noter que le désavantage de cette instruction réside dans le fait qu'**elle ne doit pas être laissée dans le code** pour ne pas apparaître et interrompre le déroulement du code lors d'un **usage ordinaire**.

APRES DEBOGAGE, N'OMETTONS PAS !

Après avoir mené à bien notre phase de débogage, **nous n'omettons pas** :

- **De supprimer tous les points d'arrêts** positionnés.
- **De supprimer tous les espions** interrompant le code.
- **De supprimer toutes les instructions « Msgbox »** introduites dans le code pour afficher des valeurs.