Les syntaxes de base

Par Maxence Hubiche

"Les fiches VBA" sont une série de petits tutoriels rapides regroupant les informations utiles sur un sujet donné. Elles concernent le langage VBA dans son ensemble et ne sont pas spécifiques à un logiciel donné. Vous y trouverez donc des informations valables pour Access, Excel, Word, Outlook, PowerPoint, ... Bonne lecture !

I - Qu'abordons-nous ici ?..... 3
II - Les syntaxes fondamentales..... 4
II-A - Les parenthèses..... 4
II-A-1 - La règle à pratiquer..... 4
II-A-2 - Les explications..... 4
II-A-3 - Cas particuliers..... 5
II-A-3-a - Le passage par Valeur..... 5
II-A-3-b - Le mot-clé "Call"..... 6
II-B - Le symbole égal (=)..... 6
II-B-1 - La règle..... 6
II-B-2 - Les explications..... 6
II-C - Les propriétés..... 7
II-C-1 - La règle..... 7
II-C-2 - Les explications..... 8
II-D - Les méthodes..... 9
II-D-1 - La règle..... 9
II-D-2 - Les explications..... 9
II-E - Les instructions et fonctions..... 11
II-E-1 - La règle..... 11
II-E-2 - Les explications..... 11
III - Remerciements..... 12

I - Qu'abordons-nous ici ?

L'objet spécifique de cette fiche est de recenser, et d'expliquer, les syntaxes de base du langage VBA.
Ce qu'il est d'ores et déjà intéressant de noter, c'est que le VBA (Visual Basic for Application) est un langage 'Basic', donc **simple** !
Le nombre de syntaxes fondamentales du langage est donc limité (5) et leur bonne connaissance sera susceptible de vous éviter bien des soucis dues aux erreurs de syntaxes.

II - Les syntaxes fondamentales

II-A - Les parenthèses

II-A-1 - La règle à pratiquer

Dans le langage VBA, on n'utilise les parenthèses QUE lorsqu'on espère obtenir un résultat (valeur ou objet) qu'on souhaite réutiliser ultérieurement

II-A-2 - Les explications

Cette règle de syntaxe, si elle est respectée, permettra de mieux comprendre le code.
Par exemple, pour commencer simplement, nous pourrions aborder la simple syntaxe de la fonction MsgBox.
En voici la syntaxe :

```
MsgBox(prompt[, buttons] [, title] [, helpfile, context])
```

On notera les points suivants :

- La syntaxe mentionne la présence de parenthèses
- La page d'aide indique qu'il s'agit d'une fonction

Pourtant, l'application stricte de la syntaxe ne fonctionne pas ! Vous pouvez essayer en écrivant le texte suivant :

```
Sub CaMarchePas()
    MsgBox ("Ceci est un essai",vbOKOnly+vbQuestion,"Syntaxe")
End Sub
```

La validation met la ligne en rouge, et, pour peu que vous ayez coché la case d'arrêt sur toutes les erreurs (dans les options), un beau message d'erreur apparaît.

Par contre, si vous n'aviez pas mis les parenthèses, le même code fonctionnerait parfaitement, et pourrait être exécuté !

```
Sub CaMarche()
    MsgBox "Ceci est un essai", vbOKOnly + vbQuestion, "Syntaxe"
End Sub
```

Cherchons à comprendre pourquoi...

Dans le premier cas, nous avons fait usage des parenthèses. Nous indiquions par cela, à l'interpréteur VBA, que nous souhaitions récupérer le résultat de la fonction MsgBox (qui, je le rappelle, est prévue, en tant que FONCTION, pour renvoyer un résultat). Mais nous avons oublié un détail d'importance ! Qu'allons nous faire de ce résultat ? Nulle part n'est indiqué l'endroit où nous allons le mettre, ce résultat. Si nous avions un tant soit peut modifié la ligne, comme ceci, par exemple :

```
Sub CaPeutMarcher()
    n= MsgBox("Ceci est un essai",vbOKOnly+vbQuestion,"Syntaxe")
End Sub
```

alors là, oui, la syntaxe aurait marché, parce que nous indiquions que nous voulions . D'ailleurs, si vous exécutez cette procédure en mode débogage (F8) vous pourrez vérifier que n récupère bien une valeur numérique en fonction du bouton sur lequel vous aurez cliqué.

Voilà pourquoi la syntaxe CaMarche fonctionne : Nous ne demandons pas la récupération d'un résultat à l'interpréteur VBA, et, par là même, nous n'avons pas besoin de le stocker nulle part.

II-A-3 - Cas particuliers

II-A-3-a - Le passage par Valeur

Certains objecteront que ma règle n'est pas vraie, car, le code suivant fonctionne :

```
Sub CaAussiCaMarche()
    MsgBox ("Ceci est un essai")
End Sub
```

On notera la mauvaise foi de ceux qui avancent ce propos, car ils n'ont mis qu'un seul argument, alors que dans les exemples que j'ai cité, il y en avait 3. déjà, rien que cela devrait nous mettre en éveil. Pourquoi, lorsqu'il n'y a qu'un seul argument, l'usage des parenthèses n'est pas contre-indiqué ?

Livrons-nous à un petit test... saisissez le code suivant dans un module

```
Sub proc_Un()           '### Procédure initiale
    Dim a As Long       'déclare la variable a comme nombre entier long
    a = 2               'affecte 2 à la variable a
    proc_Deux a         'passe a à la procédure proc_Deux
    MsgBox a            'Affiche a
End Sub

Sub proc_Deux(arg)      '### Procédure appelées
    arg = arg * 2       'multiplie l'argument par 2 et modifie sa valeur
End Sub
```

D'après vous, que vous renverra la fonction MsgBox ? Bien répondu ! vous verrez apparaître un beau "4"

Modifions maintenant un tout petit peu la procédure. C'est très léger... ajoutons des parenthèses autour du a, lors de l'appel de la proc_Deux, comme suit :

```
Sub proc_Un()           '### Procédure initiale
    Dim a As Long       'déclare la variable a comme nombre entier long
    a = 2               'affecte 2 à la variable a
    proc_Deux (a)       'passe a à la procédure proc_Deux
    MsgBox a            'Affiche a
End Sub

Sub proc_Deux(arg)      '### Procédure appelées
    arg = arg * 2       'multiplie l'argument par 2 et modifie sa valeur
End Sub
```

Et maintenant ? Que voyez-vous ? Un beau "2" !

Pourquoi ?

Parce qu'en utilisant des parenthèses autour d'un seul argument, sans demander de retourner le résultat, l'interpréteur a compris que vous passiez l'argument par VALEUR et non par REFERENCES (mais ceci fera l'objet d'une autre fiche)

Ceci nous indique donc qu'il est préférable de ne garder que la règle décrite au démarrage : n'utiliser les parenthèses que lorsqu'on souhaite obtenir un résultat qu'on exploitera ultérieurement

II-A-3-b - Le mot-clé "Call"

Le mot clé Call est un mot clé facultatif, comme vous le verrez dans la cinquième syntaxe de base du présent document. Cependant, parfois, certains programmeurs utilisent ce mot clé pour appeler, transférer le contrôle, à une autre procédure. Cette procédure peut être une procédure Sub ou une procédure Function. Mais, dans tous les cas, elle ne peut renvoyer de résultat. Du coup, on se demande quel est vraiment son usage pour les fonctions... Et puisqu'une instruction (sub) est une fonction (function) qui ne renvoie pas de résultat, on peut se demander quel en est son usage également pour les instructions. Quoi qu'il en soit, la syntaxe de Call est simple :

```
Call NomDeLaProcedure [ (ListeDesArguments) ]
```

On notera ici que les arguments - entre parenthèses, toujours - sont facultatifs. Par contre, ici, on mettra systématiquement les parenthèses, car nous sommes en présence d'une instruction qui appelle une autre instruction, et non pas une valeur ou un objet. Il est donc indispensable de "délimiter" la procédure, et cette délimitation se fera par l'usage des prenthèses. En voici un exemple :

```
Sub proc_Un()
    Dim a As Long        'déclare la variable a comme nombre entier long
    a = 2                'affecte 2 à la variable a
    Call proc_Deux(a)    'transfère le contrôle à la procédure proc_Deux, lui passant a en argument
    MsgBox a             'Affiche a (4)
End Sub
Sub proc_Deux(arg)       '### Procédure appelées
    arg = arg * 2        'multiplie l'argument par 2 et modifie sa valeur
End Sub
```

Cette fois, la boîte de message affiche bien 4. L'argument n'est pas passé par VALEUR, mais par REFERENCE, car l'usage des parenthèses est nécessaire pour délimiter la liste des arguments de proc_Deux. Si on avait voulu passer a par VALEUR, il aurait fallu ajouter une deuxième série de parenthèses :

```
Call proc_Deux((a))
```

Mais vous voyez bien que le code devient vite illisible. On évitera donc d'utiliser Call si l'on souhaite respecter la règle donnée au début de cette partie.

II-B - Le symbole égal (=)

II-B-1 - La règle

Hormis dans les tests, le symbole = est un symbole d'affectation, suivant le schéma :
Ecriture = Lecture

II-B-2 - Les explications

Le symbole égal est un symbole binaire. Il a besoin de deux opérandes ; une avant le symbole, et l'autre, après.

Le côté qui précède le symbole '=' pourrait être appelé **ECRITURE**

Le côté qui le suit serait, quant lui, appelé **LECTURE**

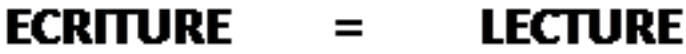
Le côté LECTURE est le côté qui est évalué. Il s'agit de la première opération qui est faite, dans l'ordre des opérations de cette ligne.

Le côté ECRITURE est le côté qui indique ce qui sera altéré, modifié, affecté. C'est la dernière opération qui est faite : l'affectation. En effet, il est difficile d'affecter une valeur (ou un objet, ou je ne sais quoi d'autre) si on ne la connait pas au préalable.

En VBA, on peut donc écrire ceci :

```
i = i + 1
```

Mathématiquement parlant, il sera extrêment difficile de démontrer cette assertion. Que se passe-t-il en fait ? Découvrez-le sur l'image gif ci-dessous :



Comment donc comprendre la phrase i=i+1 ???

- On commence par évaluer i+1
- Si i valait 10, i+1 vaut donc 11
- Nous avons la valeur de i+1
- Il faut maintenant modifier i en y mettant cette valeur
- i vaut 11

Cette ligne fait donc l'incrémentation de la variable i, d'un pas de 1, quelle qu'ait été sa valeur préalable.

II-C - Les propriétés

II-C-1 - La règle

DefinitionObjet.Propriété

II-C-2 - Les explications

Dans les écritures avec des points (.) comme, par exemple *Application.Caption*, le point indique une relation de dépendance de l'élément marqué à droite sur l'élément marqué à gauche.
Dans *Application.Caption*, on parle du Caption **DE** Application

Une propriété est une caractéristique (souvent une donnée) d'un objet.

Une propriété peut être en Lecture seule (donc, forcément à droite du =), en Ecriture seule (possible, mais quel intérêt??? donc forcément à gauche du =) ou en Lecture-écriture (donc à droite ou à gauche du symbole =)

Ces points posés, je vous propose d'essayer de répondre à cette question : Pourriez-vous me donner la couleur ?

C'est bon, vous pouvez arrêter de relire la question, elle est effectivement incomplète ! Vous ne pouvez pas y répondre, car vous vous demandez "La couleur ... de quoi ?". C'est fait exprès ! Cela pour vous montrer qu'une propriété - la caractéristique d'un objet - ne peut pas être mentionnée sans la définition de l'objet lui-même. C'est pourquoi, la syntaxe considère qu'il FAUT préciser la définition de l'objet, avant de déterminer la propriété. Mais, du coup si on définit un objet, il FAUT également préciser la propriété qui nous intéresse. On a donc forcément une syntaxe DefinitionDeLObjet.Propriété

Illustrons par quelques exemples :

Exemple Excel

```
Sheet("Feuil2").Range("A1").Formula = Sheets("Feuil1").Range("B2").Font.Name
```

Explications de cette ligne

- D'abord, on remarque le symbole = au milieu (rappel de la syntaxe précédente)
- On demande donc d'affecter à ce qui est marqué à gauche, ce qu'on va récupérer à droite.
- A droite du égal, il y a des mots séparés par des points.
- *Sheets("Feuill1").Range("B2"). Font* est la définition de l'**objet** (La Police **DE** la plage B2 (ou, en raison des parenthèses : renvoie-moi l'objet de la classe range qui s'appelle B2) **DE** la feuille Feuil1)
- *Name* est la propriété de cet objet
- On demande donc le *Nom* **DE** la *police* **DE** la *plage B2* **DE** la *feuille Feuil1*
- On en récupère la valeur (par exemple "Arial")
- Après récupération, la ligne est donc équivalente à Sheets("Feuil2").Range("A1").Formula="Arial"
- A gauche du égal, il y a aussi des mots séparés par des points.
- *Sheets("Feuil2").Range("A1")* est la définition de l'**objet** (La plage A1 **DE** la feuille Feuil2)
- Formula est la propriété de cet objet
- On veut donc modifier la Formule **DE** la *plage A1* **DE** la *feuille Feuil2*
- Puisque la propriété Formula définit ce qui est écrit dans la cellule, on veut donc écrire dans la cellule A1 de la feuille Feuil2 le nom de la police de la cellule B2 de la feuille Feuil1

Exemple Access

```
Forms("frmTest").CmdFermer.Enabled = False
```

Explication de cette ligne

- Il y a ici aussi un symbole égal, donc, nous sommes en présence d'une affectation. Ce qui est à gauche sera affecté par ce qui est à droite.
- Ce qu'on veut affecter est une valeur : FAUX
- A quoi veut-on l'affecter ? à Forms("frmTest").cmdFermer.Enabled
- Encore une écriture à points, donc, Enabled (activé) est la propriété et
- Forms("frmTest").cmdFermer est la définition de l'objet : l'objet qui s'appelle cmdFermer dans le formulaire ouvert dans Access qui s'appelle frmTest
- Il s'agit donc de rendre inactif un contrôle dans un formulaire ouvert

Exemple Word

```
ActiveDocument.Bookmarks("bkmNom").Range.text= rstContacts.Fields("ctcNom").Value
```

Explication de cette ligne

- Encore un =, donc une affectation
- Encore des . donc des definitions d'objets avec propriétés
- A gauche : La propriété Text DE la plage (range) du Signet (Bookmark) bkmNom du document actif. Voilà ce qu'on veut changer.
- A droite : La propriété Valeur du champ (field) ctcNom du Recordset rstContacts
- On va donc mettre le nom d'un contact dans un endroit prédéfini du document, à l'emplacement du signet.

On pourrait continuer comme cela encore longtemps, mais je supposerai que vous avez compris arrivé à ce niveau de l'explication.

II-D - Les méthodes

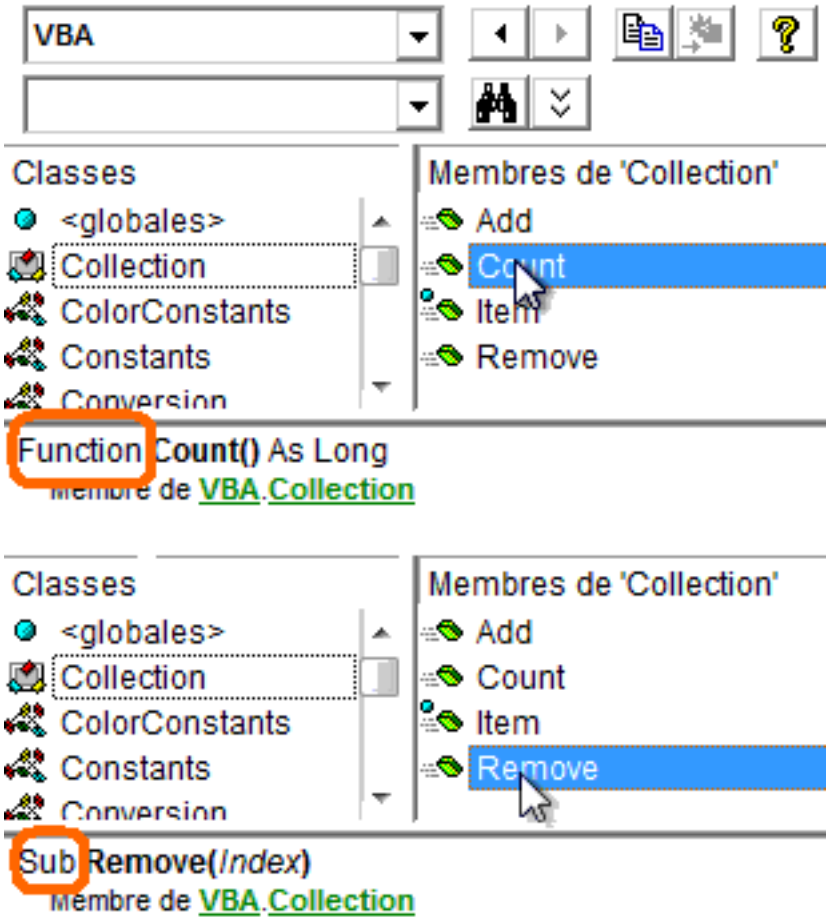
II-D-1 - La règle

DefinitionObjet.Méthode [()[ListeArguments][)]

II-D-2 - Les explications

Dans les écritures avec des points (.) comme, par exemple *Range("B2").Select*, le point indique une relation de dépendance de l'élément marqué à droite sur l'élément marqué à gauche.
Dans *Range("B2").Select*, on parle de la méthode Select **qui s'applique à** Range("B2")

Une méthode est une action qu'on exécute avec, ou à partir de l'objet. Techniquement parlant, il s'agit d'une sub ou d'une fonction écrite dans un module de classe. Vous pouvez le voir dans l'explorateur d'objets (F2). L'exemple ci-dessous vous montre deux méthodes de la classe Collection de la bibliothèque VBA. Notez que l'une est une Fonction (Function) alors que l'autre est une Instruction (Sub) :



Donc, si la méthode est une fonction, et que nous voulons en récupérer le résultat, nous mettrons les parenthèses de la syntaxe. dans les autres cas, les parenthèses ne sont pas à mettre.

Quant aux arguments, il y a des actions qui nécessitent une information pour être exécutées. D'autres non. C'est la raison d'être des arguments.

Illustrons par quelques exemples

Exemple Excel

```
Range("A1:D1").BorderAround xlDouble, , 3
```

Explication de cette ligne de code

- Pas de symbole égale, donc pas d'affectation
- Ecriture avec des points, donc méthode
- La méthode : Appliquer une bordure au contour (BorderAround) de la Plage allant de A1 jusqu'à D1
- Les arguments :
- >> le premier est le style de bordure
- >> le deuxième est sauté (facultatif)
- >> le troisième est la couleur de la bordure
- Les arguments ainsi définis permettent de définir quel genre de bordure sera appliqué sur le contour de la plage.

Exemple Access/DAO

```
CurrentDb.QueryDefs.Delete "qryAVirer"
```

Explication de cette ligne de code

- Pas de symbole = et écriture en point. Nous sommes en présence d'une méthode.
- La méthode : Delete
- L'objet : La collection des requêtes (QueryDefs) de la base de données en cours (CurrentDB)
- L'argument :
- >> Le nom de la requête à supprimer dans la collection (qryAVirer)
- Cette ligne permet donc de supprimer une requête définie parmi toutes celles qui sont dans la base de données en cours.

Exemple Word

```
ActiveDocument.Paragraphs(i).Range.Select
```

Explication de la phrase

- Une phrase avec des points, et pas de symbole =. Nous sommes en présence d'une méthode
- Pas d'espaces. Il n'y a pas d'arguments
- La méthode : Select
- L'objet : La plage (range) du ième paragraphe (Paragraph) du document actif (ActiveDocument)
- Cette phrase sert donc à sélectionner toute la zone du ième paragraphe du document en cours.

II-E - Les instructions et fonctions

II-E-1 - La règle

NomProcedure [([ListeArguments])]

II-E-2 - Les explications

La boucle est bouclée. Puisqu'une méthode est une Sub ou une Function écrite dans un module de classe, la syntaxe pour les instructions et fonctions (Sub et Function dans des modules standards) est très proche ! La seule différence étant l'objet. il n'y en a plus, puisqu'il n'y a plus de classe !

Comparez donc la syntaxe pour les méthodes avec celle des instructions et procédures, et vous verrez que la seule différence, c'est que l'objet a disparu.

Donc, tout est dit !

- Pas besoin de **Call**
- Les parenthèses seulement s'il s'agit d'une fonction dont on veut récupérer le résultat
- Utilisation du = si l'on souhaite affecter le résultat à quelque chose

III - Remerciements

Je tiens à remercier le travail d'équipe de tous les bénévoles qui participent à ce site, et, dans le cadre des observations et corrections apportées à ce tutoriel, mes remerciements vont tout particulièrement à

Correction

- **Dolphy35**
- **Arkham46**
- **Heureux-Oli**
- **Silkyroad**

Commentaires

- **Bidou** - mais je ne sais pas si je devrais le mettre dans les remerciements ... va falloir que je me tape un tuto sur le scope maintenant ... pfff :)