Initiation au VBA Office

par Olivier Lebeau (Heureux-oli sur DVP)

Date de publication : 24 janvier 2009

Dernière mise à jour :

Cet article a pour but de jeter les bases du langage VBA. Il sera complété par d'autres articles spécifiques aux produits de la suite Office.

1 - Introduction....4
2 - L'enregistreur de macro....4
2-A - L'enregistreur de Macro Word....4
2-B - L'enregistreur de Macro Excel....7
2-C - PowerPoint....11
2-D - Access et Outlook....11
3 - Le VBE....12
3-A - Word....12
3-B - Excel....15
4 - Modification de notre code....16
5 - Les variables, durée de vie et portée....20
5-A - Dim....20
5-B - Static....22
5-C - Private....22
5-D - Public....22
6 - Constantes....22
7 - Type de données utilisateur....23
8 - Les procédures....23
8-A - Les Sub....23
8-A-1 - Portée d'une Sub....23
8-B - Les Function....25
8-B-1 - Portée d'une Function....25
8-B-2 - Valeur renvoyée par une fonction....25
8-C - Appel d'une procédure....26
8-C-1 - ByVal ou ByRef....28
8-C-2 - Optional....29
9 - Écriture de votre code....29
9-A - Les commentaires....29
9-B - Les retours à la ligne....30
9-C - L'indentation....30
10 - Les opérateurs....30
10-A - Affectation (=)....30
10-B - Arithmétiques....31
10-C - Comparaison....32
10-D - Concaténation....33
10-E - Opérateurs logiques....33
10-E-1 - Or....33
10-E-2 - And....34
10-E-3 - XOR....35
10-E-4 - NOT....36
11 - Les structures décisionnelles....37
11-A - La structure If ... Then 37
11-B - La structure If ... Then ... ElseIf ... Else... End If....37
11-C - La structure Select Case....38
12 - Les boucles....39
12-A - Do While .... Loop....39
12-B - While .... Wend....39
12-C - Do Until .... Loop....40
12-D - Do .... Loop While....40
12-E - Do ... Loop Until....40
12-F - For ... To .... Next....41
12-G - For Each .... Next 41
13 - Quelques fonctions VBA....42
13-A - Fonctions Texte....42
13-A-1 - Len....42
13-A-2 - UCase....42
13-A-3 - LCase....43
13-A-4 - Left....43

13-A-5 - Right....43
13-A-6 - Mid....43
13-A-7 - Chr....44
13-A-8 - Asc....44
13-A-9 - Trim - LTrim - RTrim....46
13-A-10 - StrReverse....46
13-A-11 - Replace....46
13-A-12 - InStr....47
13-A-13 - Split....47
13-B - Fonctions de conversion....48
13-B-1 - Str....49
13-C - Fonctions Date....49
13-C-1 - Date()....49
13-C-2 - Now()....49
13-C-3 - DateAdd....50
13-C-4 - DateDiff....51
13-C-5 - DatePart....52
13-C-6 - DateSerial....53
13-C-7 - DateValue....53
13-D - Fonctions Système....53
13-D-1 - Beep....53
13-D-2 - Environ....54
13-D-3 - DoEvents....54
13-D-4 - CreateObject....54
13-D-5 - GetObject....54
13-E - InputBox - MsgBox....55
14 - Liens Utiles....55
15 - Remerciements....55

1 - Introduction

Cet article s'adresse aux débutants en VBA. Si vous avez déjà utilisé le VBA, vous allez vous ennuyer en lisant ces lignes. Par contre, si vous voulez découvrir un autre moyen d'utiliser les logiciel de la suite Office, j'espère que vous apprendrez quelque chose.

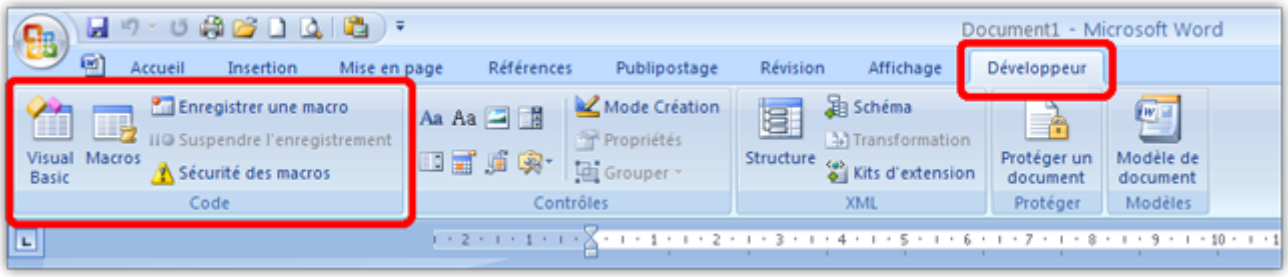
Pour faire du VBA, vous avez deux options, la première est l'écriture du code dans son intégralité et la seconde l'utilisation de l'enregistreur de Macro fourni avec certains logiciels de la suite Office.

2 - L'enregistreur de macro

2-A - L'enregistreur de Macro Word

L'"**Enregistreur de Macro**" est un outil fourni avec Word pour vous permettre d'utiliser le VBA sans pour autant connaître le langage. Nous allons utiliser l'enregistreur pour ensuite visualiser le code qui aura été enregistré.

Pour activer l'enregistreur de macro, nous avons au moins deux possibilités. La première est de passer par l'onglet développeur et la seconde par la barre d'état de Word.
Il en existe une troisième par l'**Onglet Affichage** et dans le groupe **Macro**.



L'onglet Développeur

Dans la barre d'état

*Si l'onglet developpeur n'est pas visible, pour l'afficher : Bouton **Office** -> **Options Word** -> **Standard** -> Afficher l'onglet développeur dans le Ruban.*

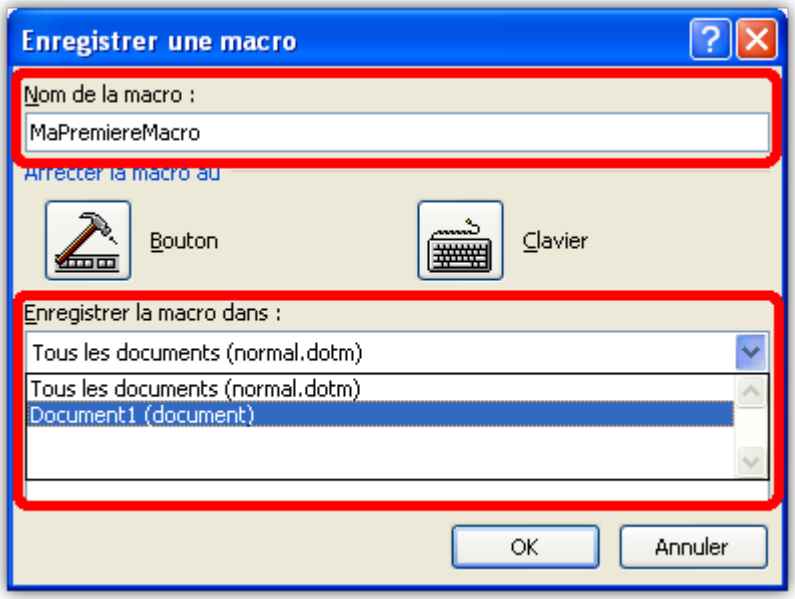
Le résultat reste le même et ne dépend pas du choix fait.

Si vous travaillez sous Word 2003, vous avez accès aux macros par le menu **Outils** -> **Macro** et Enregistreur de Macro. La suite ne change pas.

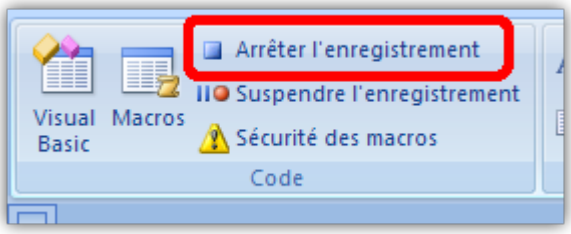
Nous allons commencer par changer le nom donné par défaut "**Macro1**" par "**MaPremiereMacro**". Même si pour VBA ça n'a aucune importance, je vous conseille d'éviter les espaces et les accents dans les noms que vous allez donner à vos macros. Pour les espaces, Word vous enverra sur les roses.

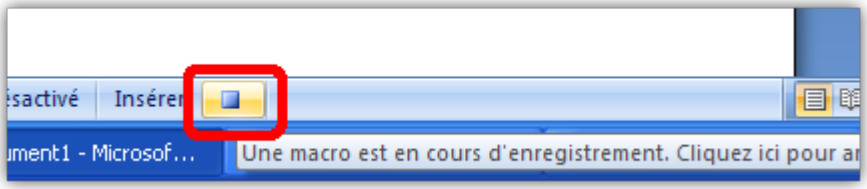
Si vous voulez avoir plus de clarté dans les noms, utilisez des majuscule en début de mot vos nom seront plus lisibles, que ce soit pour une variable, une macro ou tout autre nom. MaPremiereMacro est plus lisible que mapremieremacro.

Cette boîte de dialogue vous propose de mettre la macro dans le normal.dotm, ça ne pose pas de problème, mais pour le début, nous allons utiliser le document vierge que nous recevons lors de l'ouverture de Word. Si un autre document est ouvert, vous aurez un choix supplémentaire.

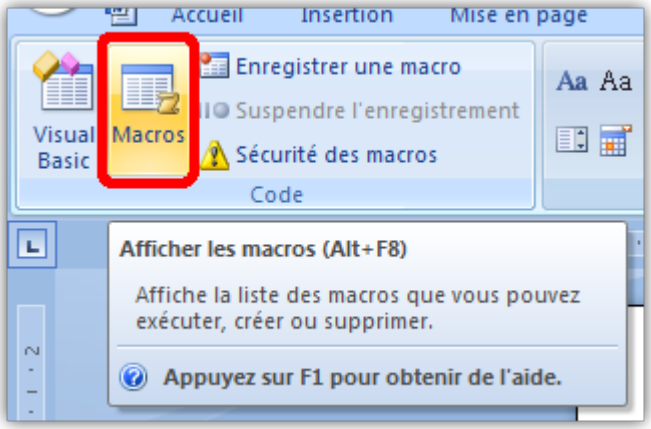


Dès que vous avez cliqué sur OK, l'enregistreur commence sont travail, toutes les actions que vous allez effectuer seront enregistrées.
Vous allez écrire un texte dans votre document.
Ma première macro.
Lorsque vous avez terminé, n'oubliez pas d'arrêter l'enregistrement.



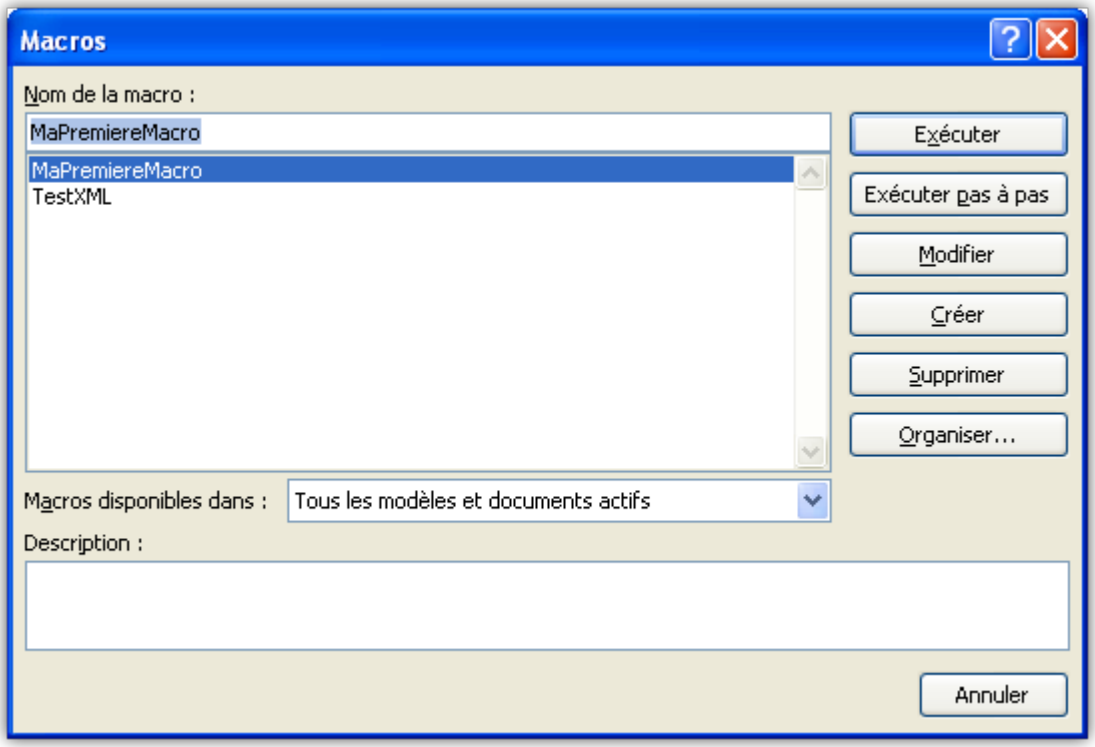


Supprimez la ligne que vous venez d'écrire dans votre document et cliquez sur le bouton Macro.



Le bouton Macro

Vous ouvrez alors une boîte de dialogue contenant le nom de votre Macro.

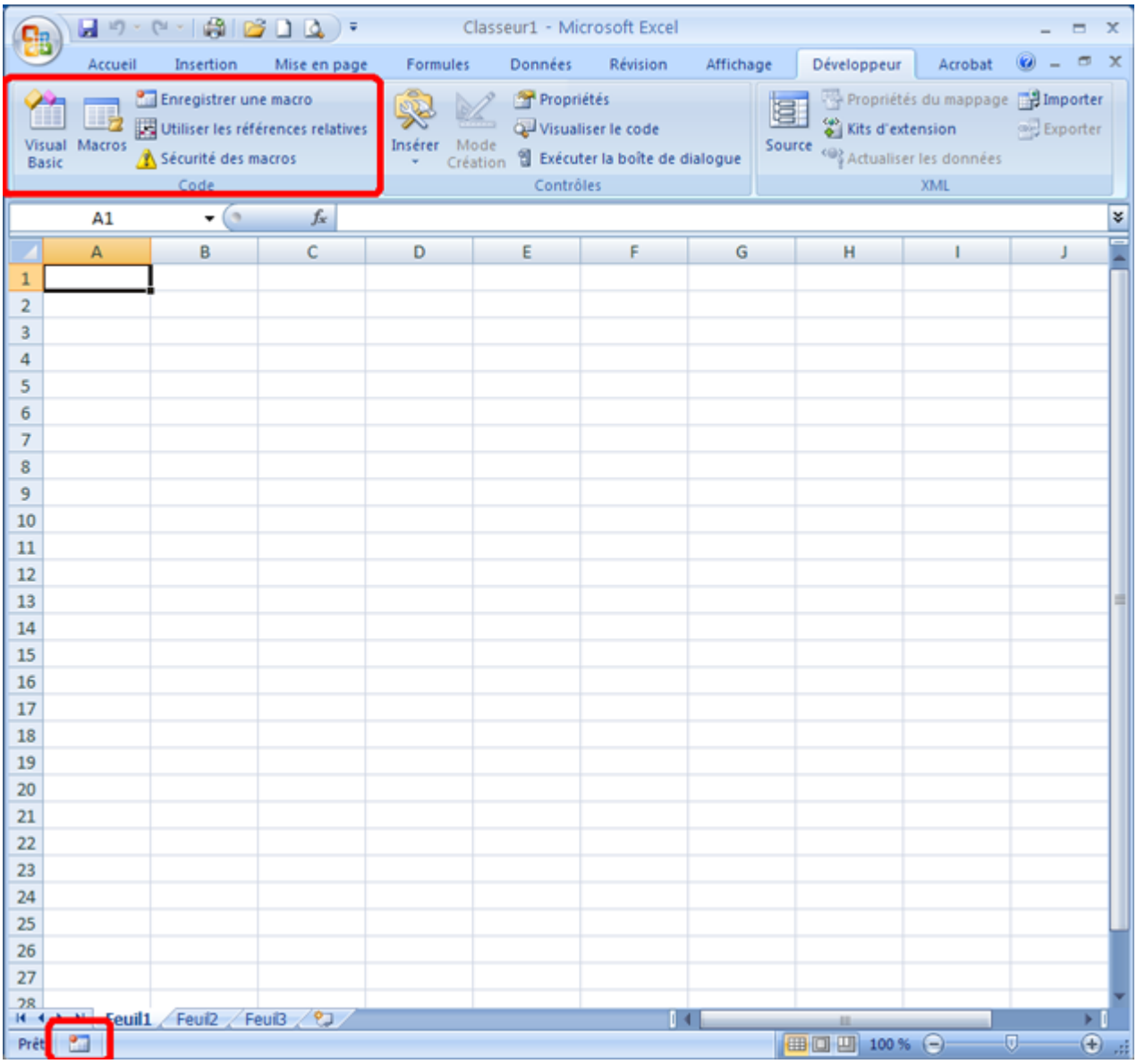


La liste des macros disponibles

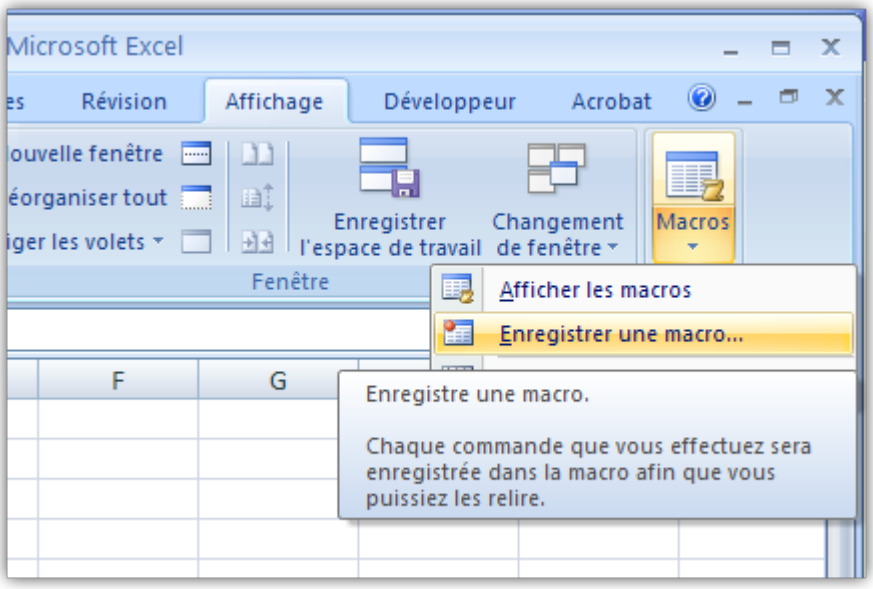
Exécutez la macro.
Le texte que vous venez de supprimer est à nouveau présent sur le document, vous pouvez répéter cette manipulation autant de fois que vous le souhaitez.

2-B - L'enregistreur de Macro Excel

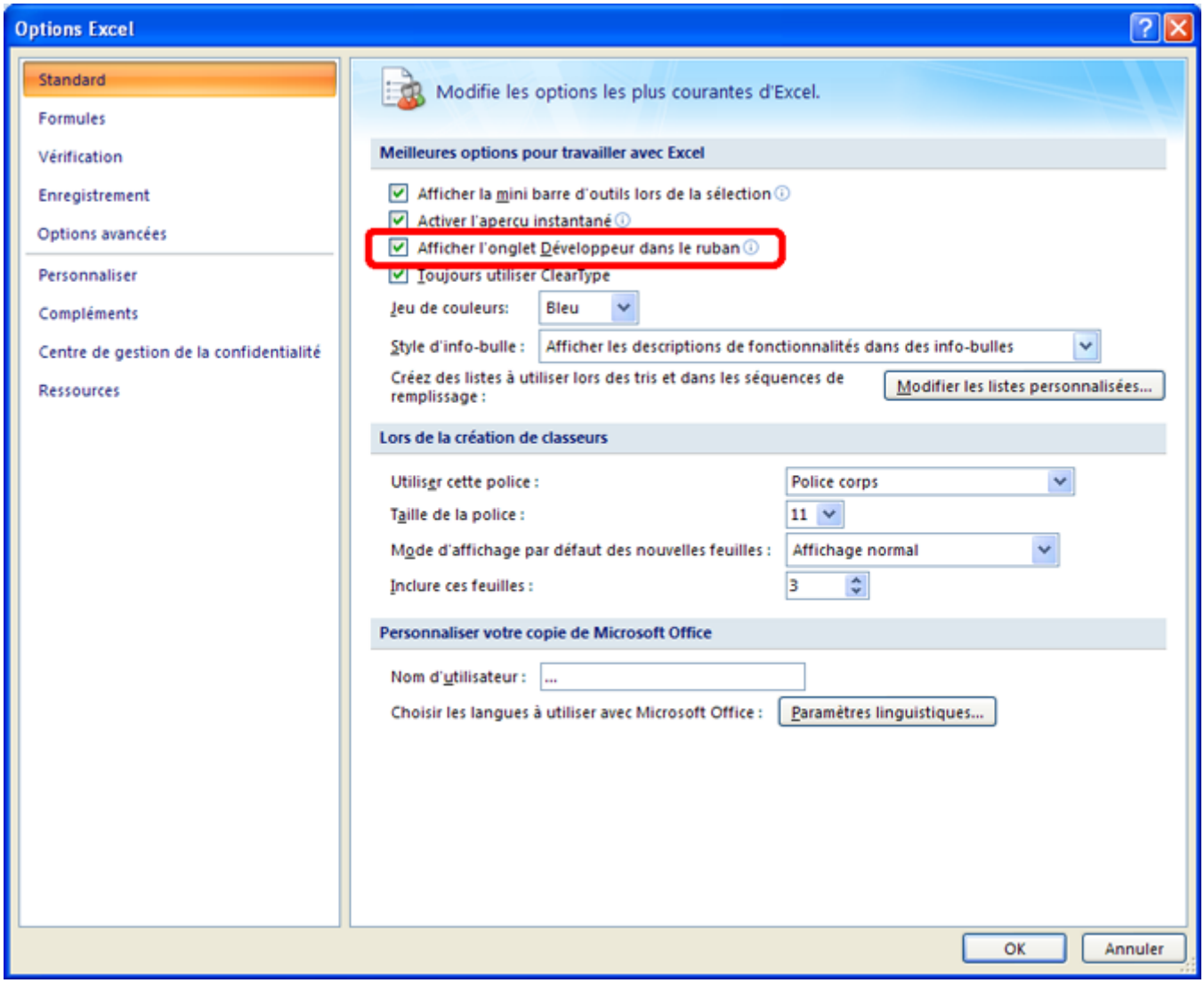
Comme pour Word, Excel est fourni avec un enregistreur de macro. Pour activer l'enregistreur, vous avez trois possibilités.
Via l'onglet **Développeur** et le bouton **Enregistrer une macro**, via la barre d'état et le bouton des macros.



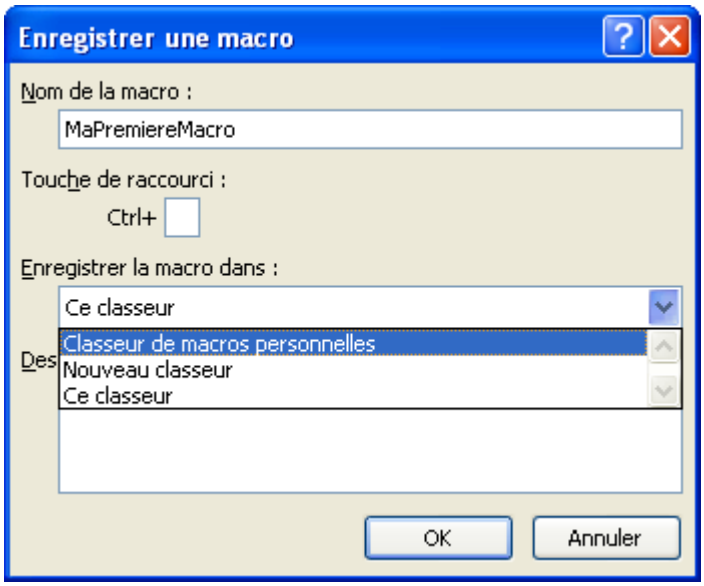
Et finalement, via l'onglet **Affichage** et le déroulant relatif aux Macros.



Si l'onglet développeur n'est pas visible, pour l'afficher, Bouton Office => Standard => Afficher le bouton développeur dans le ruban.



Si vous travaillez aec Excel 2003, vous trouverez l'enregistreur de macro dans **Outils** => **Macro**.

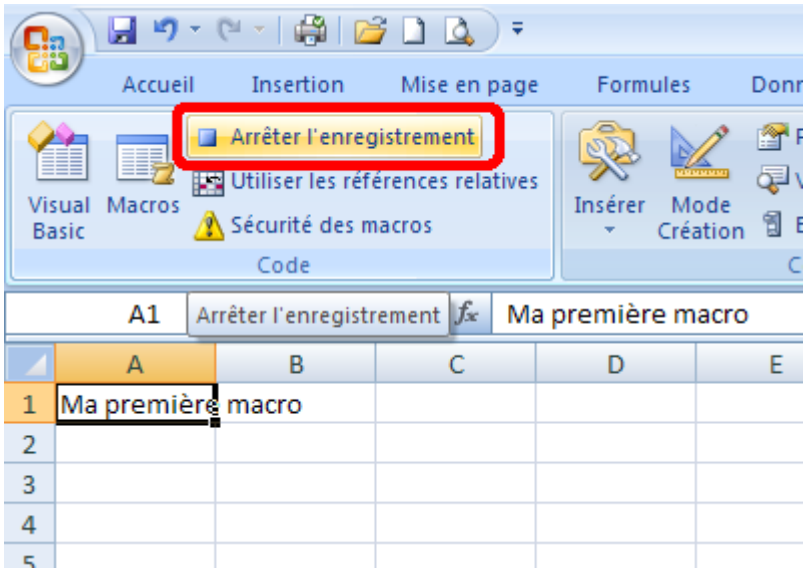


Vous ouvrez une boîte de dialogue vous permettant de renseigner quelques informations.
La première est le nom de la macro, vous allez saisir "**MaPremiereMacro**" dans ce nom, vous ne pouvez pas avoir d'espace, les accents sont autorisé, mais par facilité, je vous coinseille vivement de ne pas les utiliser.

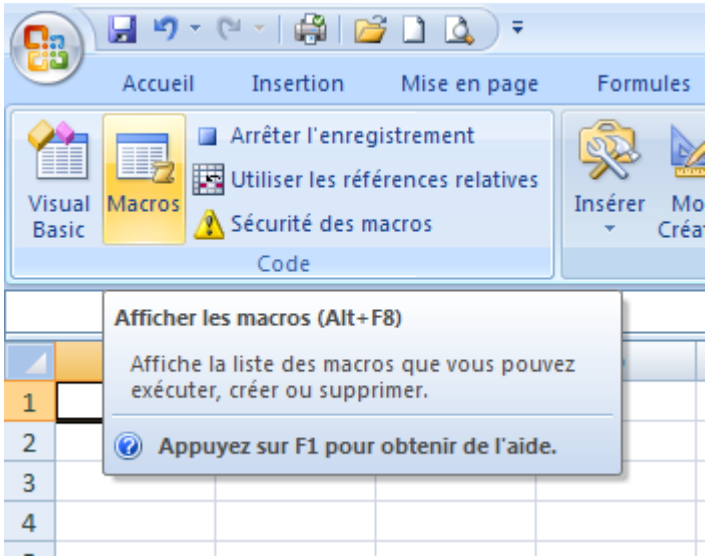
Si vous voulez avoir plus de clarté dans les noms, utilisez des majuscule en début de mot vos nom seront plus lisibles, que ce soit pour une variable, une macro ou tout autre nom. MaPremiereMacro est plus lisible que mapremieremacro.

Cette boîte de dialogue vous propose de mettre votre macro dans le classeur actif, dans le **Classeur de macros personnelles** ou dans un nouveau classeur. Si vous mettez votre macro dans le **Classeur de macros personnelles**, elle sera disponible pour tous vos classeurs. Nous allons la mettre dans le classeur actif "**Ce classeur**".

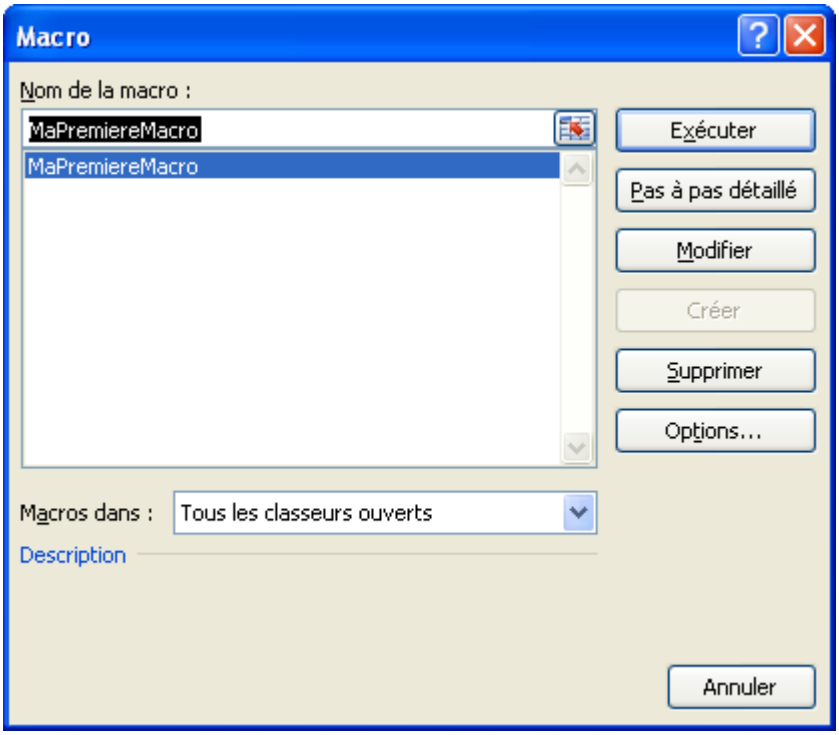
Dès que vous avez cliqué sur Ok, votre macro est en cours d'enregistrement, toutes les actions que vous allez faire seront enregistrées. Pour cette première macro, je vous invite à écrire dans la première cellule (A1) de votre feuille : "Ma première macro". Quittez la cellule et arrêtez l'enregistrement de votre macro.



Supprimez le texte que vous venez d'écrire dans la cellule. Lancez la macro que vous venez de faire :



Choisissez votre macro :



Lancez l'exécution et votre texte va réapparaître dans la cellule. Vous pouvez utiliser cette macro autant de fois que vous le souhaitez.

2-C - PowerPoint

L'accès à l'enregistreur de macro n'est pas direct pour PowerPoint 2007. Cependant, il existe une compatibilité de commandes entre 2003 et 2007 ce qui nous permet de lancer en mode aveugle l'enregistreur de macro avec la combinaison de touches **Alt O M E**.

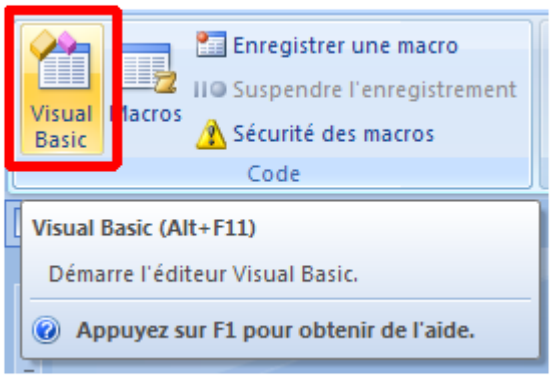
2-D - Access et Outlook

Malheureusement, ces deux logiciels ne possèdent pas d'enregistreur de Macro. Ces étapes ne pourrons pas être utilisées.

3 - Le VBE

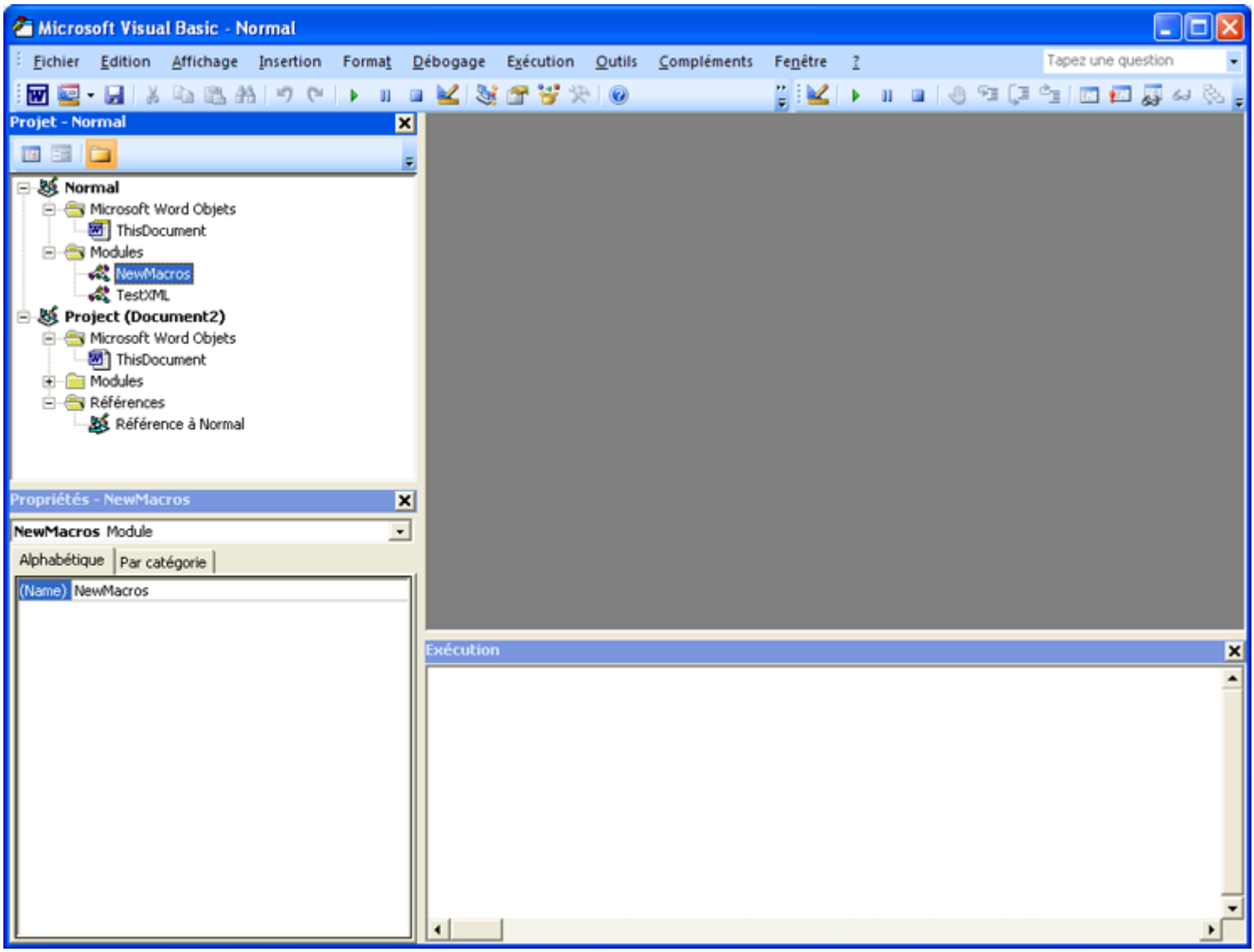
3-A - Word

Ces trois lettres sont les abréviations **V**isual **B**asic **E**ditor. Cette interface graphique vous permet de visualiser, éditer ou insérer du code. Pour y accéder, vous avez le choix, personnellement, j'utilise la combinaison de touches *Alt + F11*, ou utiliser le bouton du Ruban.

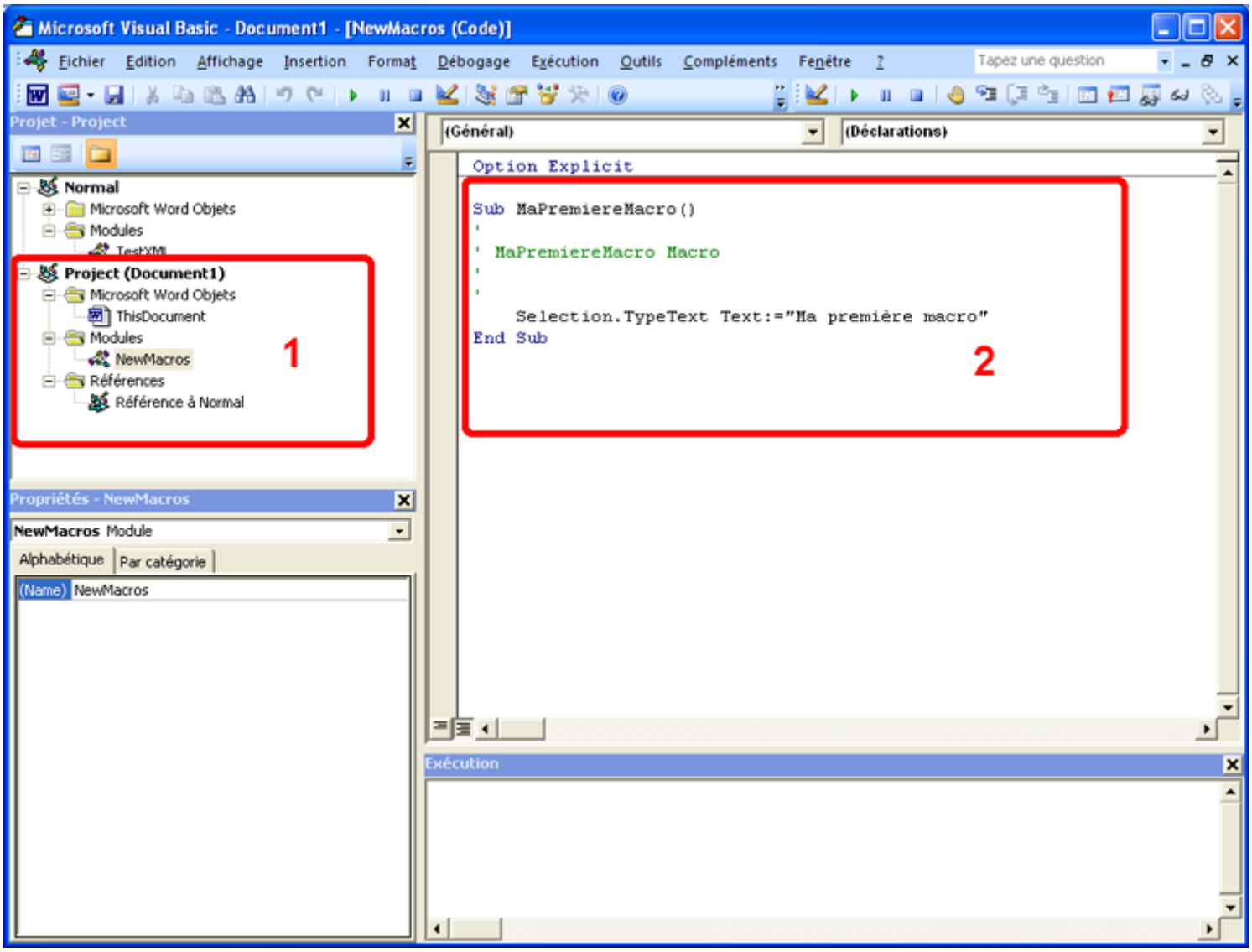


Le bouton du Ruban

Lorsque vous ouvrez le VBE, vous obtenez ceci :



Pour l'instant, nous allons nous attarder sur deux parties de notre VBE.



La première partie (1) donne une arborescence avec le contenu de votre projet qui se compose de :
Microsoft Word Objets composé de **ThisDocument**
Module contenant un module **NewMacros**
Référence intégrant **Référence à Normal**.

Nous allons double cliquer sur le module **NewMacros**, c'est dans ce module que se trouve la macro que nous venons d'enregistrer. Lorsque nous double cliquons sur ce module, nous affichons son contenu dans la fenêtre d'édition.

La seconde partie (2) est la zone d'édition qui contient notre code.

```
Sub MaPremiereMacro()
'
' MaPremiereMacro Macro
'
'
    Selection.TypeText Text:="Ma première macro"
End Sub
```

Si nous détaillons ce code, nous avons quatre lignes.
La première et la dernière ligne vont de pair.

```
Sub MaPremiereMacro()
 ....
 ....
End Sub
```

La première constitue le début de notre procédure, elle contient le nom de notre macro.
La dernière est la fin de notre procédure, le code est contenu entre ces deux balises.

```
'
' MaPremiereMacro Macro
'
'
```

Cette ligne constitue un commentaire. Les commentaires en VBA commencent toujours par une apostrophe ('). Les lignes qui suivent ce caractère sont ignorées lors de l'exécution du code.

```
Selection.TypeText Text:="Ma première macro"
```

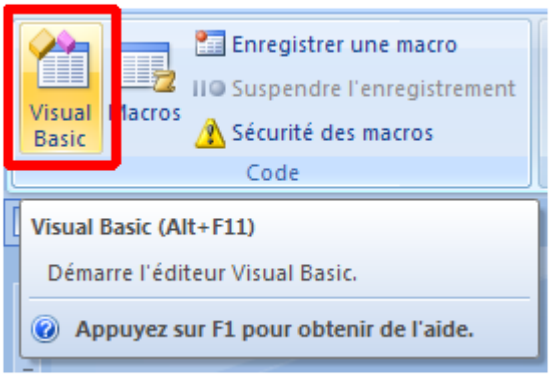
Nous terminons par notre ligne de code. Dans cet exemple, elle est très simple. Nous utilisons un Objet et sa méthode et l'argument utilisé pour la méthode.

- Objet : Selection
- Méthode : TypeText
- Argument : Text:="Ma première macro"

Vous pouvez réutiliser ce code un très grand nombre de fois.
Vous venez de faire votre première **Macro**.

3-B - Excel

Cette partie est similaire à ce que nous venons de voir pour Word. Au lieu d'avoir des documents, nous avons des Classeurs contenant des feuilles. La macro que nous venons d'enregistrer se trouve dans le Module1 qui a été créé par Excel pour recevoir votre macro.




```
Sub MaPremiereMacro()
'
' MaPremiereMacro Macro
'

'
    ActiveCell.FormulaR1C1 = "Ma première macro"
```

```
    Range("A2").Select

End Sub
```

Dans notre code, nous avons cinq lignes.

```
Sub MaPremiereMacro()
...
...
End Sub
```

La première ligne constitue le début de notre procédure. La dernière, la fin de notre procédure, le code est toujours contenu entre ces deux lignes.

```

'
' MaPremiereMacro Macro
'
```

Cette ligne constitue un commentaire, les commentaires en VBA commencent toujours par une ', les lignes qui suivent seront ignorées lors de l'exécution du code.

```

    ActiveCell.FormulaR1C1 = "Ma première macro"
    Range("A2").Select
```

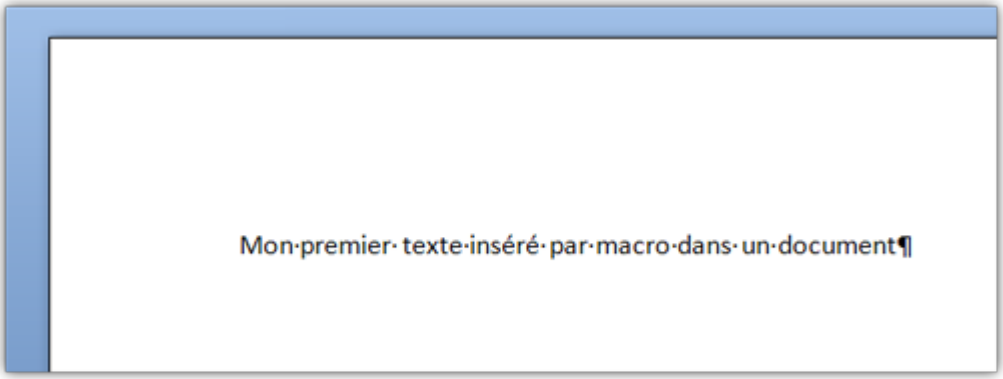
Nous terminons par nos deux lignes de code, la première va insérer le texte "Ma première macro" dans la cellule active et la seconde va activer la cellule A2.

4 - Modification de notre code

En Word

```
Sub MaPremiereMacro()
'
' MaPremiereMacro Macro
'
'
 Selection.TypeText Text:="Ma première macro"
End Sub
```

Si nous modifions le code obtenu en modifiant "Ma première macro" par "Mon premier texte inséré par macro dans un document". Nous lançons l'exécution de la macro. Si vous n'avez pas commis d'erreur, vous devriez avoir ceci :

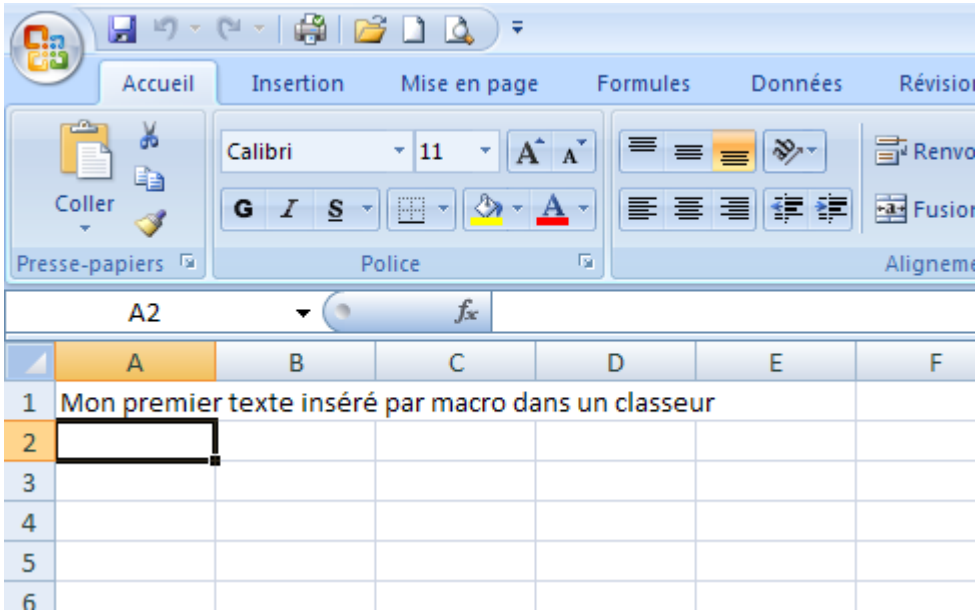


En Excel

Effectuons la même manipulation en Excel, sur le code que nous avions obtenu.

```
Sub MaPremiereMacro()
'
' MaPremiereMacro Macro
'
    ActiveCell.FormulaR1C1 = "Ma première macro"
    Range("A2").Select


End Sub
```

Si nous modifions le code obtenu en modifiant "Ma première macro" par "Mon premier texte inséré par macro dans un classeur". Nous lançons l'exécution de la macro. Si vous n'avez pas commis d'erreur, vous devriez avoir ceci :



La première conclusion est que le texte situé entre " est intégré à notre document ou classeur. Dans l'état actuel des choses, ce n'est pas très pratique, chaque fois que ce texte va changer, vous devrez modifier la macro. Si c'est une autre personne qui utilise votre document, elle devra changer ce texte dans le code, si votre code comporte plusieurs centaines de lignes, il sera nécessaire de trouver la ligne à modifier.

Pour palier ces inconvénients, il existe une solution simple : la variable.
Les variables servent à stocker des données pour pouvoir être utilisées ou manipulées. Une variable est un endroit de la mémoire où sera stockée la donnée. Vous pouvez utiliser la variable une seule fois ou un grand nombre de fois.

Il existe différents types de variable qui dépendent de la donnée que la variable va recevoir. Ci dessous la liste des types de variables que vous pouvez utiliser en VBA.

Type	Utilisation
Boolean	Deux valeurs possible : 0 ou 1; Vrai ou Faux
Byte	Nombre de 0 à 255
Integer	Nombre de -32 768 à 32 767
Long	Nombre de -2 147 483 648 à 2 147 483 647
Single	Nombre de -3,402823E38 à -1,401298E-45 pour les valeurs négatives et de 1,401298E-45 à 3,402823E38 pour les valeurs positives
Double	Nombre de -1,79769313486231E308 à -4,94065645841247E-324 pour les valeurs négatives et de 4,94065645841247E-324 à 1,79769313486232E308 pour les valeurs positives
Currency	Nombre de -922 337 203 685 477,5808 à 922 337 203 685 477,5807
Decimal	+/-79 228 162 514 264 337 593 543 950 335 sans séparateur décimal ; +/-7,9228162514264337593543950335 avec 28 chiffres à droite du séparateur décimal ; le plus petit nombre différent de zéro est +/-0.0000000000000000000000000001.
Date	Une date du 1er janvier 100 au 31 décembre 9999 Une date est comparable à un double.
Object	Toute référence à des données de type Object Un paragraphe est un objet
String (longueur variable)	Chaîne de 0 à environ 2 milliards de caractères
String (longueur fixe)	Chaîne de 1 à environ 65 400 de caractères
Variant	Tout nombre = 16 octets pour un nombre Chaîne = 22 octets + longueur de la chaîne pour un texte

L'utilisation d'une variable est assez simple, on commence par la déclarer pour ensuite lui affecter une valeur.

```
Sub MaPremiereMacroAvecVariable()
'Nous allons déclarer une variable de type string
'Elle va recevoir du texte.
Dim strMaVar As String
...
...
...
End Sub
```

La déclaration d'une variable commence dans la majorité des cas par Dim.
Dim signifie Dimension, on va réserver un espace en mémoire pour la donnée.
Nous verrons plus loin qu'il existe d'autres moyens de déclarer les variables.

```
Sub MaPremiereMacroAvecVariable()
'Nous allons déclarer une variable de type string
'Elle va recevoir du texte.
Dim strMaVar As String
'Affectation d'un valeur
strMaVar = "Bonjour à tous !"
```

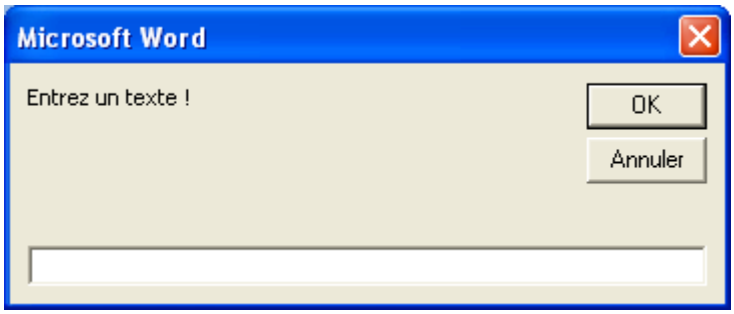
```
...
...
End Sub
```

Pour affecter une valeur à la variable, nous allons simplement utiliser l'opérateur d'affectation "=". La variable contient maintenant comme donnée "Bonjour à tous !".

Pour illustrer l'utilisation des variables nous allons utiliser une partie du code que nous avons obtenu avec l'enregistreur de macro et faire appel à une petite fonction "**InputBox**" pour dialoguer avec l'utilisateur.

```
Sub MaPremiereMacroAvecVariable()
'Nous allons déclarer une variable de type string
'Elle va recevoir du texte.
Dim strMaVar As String
'Affectation d'un valeur
strMaVar = InputBox("Entrez un texte !")
 Selection.TypeText Text:=strMaVar
End Sub
```

Lancez l'exécution de ce code et si vous n'avez pas commis d'erreur, vous devriez voir apparaitre une boîte de dialogue où vous saisirez le texte qui sera introduit dans votre document.



Ce qui s'est produit, le texte que vous avez entré dans la boîte de dialogue a été affecté à la variable.
strMaVar = InputBox("Entrez un texte !")
La variable a ensuite été utilisée pour insérer le texte dans le document.
Selection.TypeText Text:=strMaVar Ce n'est donc pas strMaVar qui a été inséré, mais la donnée qu'il contient.

Le texte de la variable est-il à nouveau utilisable ?

5 - Les variables, durée de vie et portée

Nous avons terminé le chapitre précédent sur une question. La réponse est non dans l'exercice que nous avons fait. En fonction de la méthode utilisée et de l'endroit où sont déclarées les variables, elles n'ont pas les mêmes caractéristiques.

5-A - Dim

Dim est l'attribut le plus fréquent pour la déclaration des variables. Lorsque Dim est utilisé à l'intérieur d'une procédure, la variable est disponible pour cette procédure et n'aura de temps de vie que pendant le déroulement de la procédure

```
Sub MaPremiereMacroAvecVariable()
'Nous allons déclarer une variable de type string
'Elle va recevoir du texte.
Dim strMaVar As String
'Affectation d'un valeur
strMaVar = InputBox("Entrez un texte !")
```

```
 MsgBox strMaVar
End Sub
```

Dans les lignes de code qui précèdent, c'est le cas, la variable n'existe plus lorsque que le code est terminé.
Nous allons faire un petit test en ajoutant sous ces lignes de code une seconde procédure mais sans déclarer la variable.

```
Sub TestDeVie()
MsgBox strMaVar
End Sub
```

Lancez l'exécution des deux procédures l'une après l'autre.
Lors du lancement de la seconde, vous recevrez un petit message : "Erreur de compilation : Variable non définie".
Notre variable n'existe plus.

Autre petite expérience, déplacez la déclaration de la variable hors de la procédure.

```
Option Explicit

'Nous allons déclarer une variable de type string
'Elle va recevoir du texte.
Dim strMaVar As String

Sub MaPremiereMacroAvecVariable()

'Affectation d'un valeur
strMaVar = InputBox("Entrez un texte !")
    MsgBox strMaVar
End Sub

Sub TestDeVie()
MsgBox strMaVar
End Sub
```

Lancez l'exécution des deux procédures l'une après l'autre.
Vous ne recevez plus de message d'erreur, mais en plus la variable ne perd pas sa valeur.
Essayons cette fois d'ajouter un module à notre projet.
Insertion -> **Module**
Pour accéder au module, dans l'explorateur de projet, vous avez maintenant un répertoire supplémentaire "Modules", dans ce répertoire, vous avez votre nouveau module. Double cliquez sur ce module pour pouvoir éditer le code qu'il contient ou ajouter du code s'il n'en contient pas. Dans ce module, collez la dernière procédure et ajoutez "_1" au nom de la procédure.

```
Sub TestDeVie_1()
MsgBox strMaVar
End Sub
```

Exécutez à nouveau le code de la procédure "**MaPremiereMacroAvecVariable**".
Changez de module et exécutez la procédure "**TestDeVie1**". Vous recevez un message désagréable : "Erreur de compilation : Variable non définie".

Résumé

Lorsqu'une variable est déclarée avec le mot Dim dans une procédure, elle est utilisable dans cette procédure.
Lorsqu'elle est déclarée avec le mot Dim dans un module, elle n'est utilisable que par les procédures de ce module.

5-B - Static

Nous avons vu que lorsqu'une variable est déclarée dans une procédure avec l'attribut Dim, la valeur contenue est supprimée à la fin de la procédure.
Si vous ne voulez pas déclarer la variable hors de votre procédure, vous pouvez utiliser l'attribut "**Static**" et la valeur contenue dans la variable sera conservée entre deux exécutions.

```
Sub MaPremiereMacroAvecVariable()
'Nous allons déclarer une variable de type entier

Static intMaVar As Integer
intMaVar = intMaVar + 1
MsgBox "Ma variable vaut : " & intMaVar

End Sub
```

5-C - Private

Vous pouvez déclarer une variable avec l'attribut **Private**. La déclaration avec Private ne se fait qu'au niveau module. Une variable déclarée avec Private n'est utilisable que par les procédures contenues dans le module.

Vous ne pouvez pas utiliser Private dans une procédure.

```
Private objDoc As Document

Sub NouveauDocument()
Set objDoc = Application.Documents.Add
End Sub
```

5-D - Public

Vous pouvez déclarer une variable avec l'attribut Public. La déclaration d'une variable avec Public ne peut se faire qu'au niveau du module et est utilisable par toutes les procédures de votre projet.

```
Public objDoc As Document

Sub NouveauDocument()
Set objDoc = Application.Documents.Add
End Sub
```

6 - Constantes

Les constantes sont déclarées à l'aide de l'attribut Const. Elles doivent être déclarées dans un module simple et pas dans un module de classe.

```
Const RacCar2 As Double = 1.414213562
Sub Carre()
Debug.Print RacCar2 * RacCar2

End Sub
```

La constante reçoit sa donnée dès la déclaration et ne pourras plus être modifiée.

7 - Type de données utilisateur

Ce type de données ne peut pas être déclaré dans un module de classe. Évitez donc le module ThisDocument.

Ce type de données est déclaré avec le mot réservé **Type** les attributs Public et Private ont le même effet sur une variable de type utilisateur que sur les autres variables.

Type de données utilisateur

```
Public Type MyVar
    IntA As Integer
    IntB As Integer
End Type
```

Voyons maintenant comment mettre ce type de données en oeuvre.

```
Public Type MyVar
    IntA As Integer
    IntB As Integer
End Type
'*******************************************************
Sub MaProc()
Dim MyData As MyVar

With MyData
    .IntA = 10
    .IntB = 20
End With

MsgBox  MyData.IntA & " - " & MyData.IntB

End Sub
```

Pour affecter des données à un type personnel, après déclaration, on fait suivre le nom de la variable par un point (.) et le nom de l'élément.

8 - Les procédures

8-A - Les Sub

Une **Sub** est une procédure qui ne renvoie pas de résultat. Elle commence toujours par **Sub** et se termine par **End Sub**. Comme pour les variables, une procédure a une portée qui dépend de la manière dont elle est déclarée.

8-A-1 - Portée d'une Sub

Une **Sub** peut être déclarée en **Public** ou en **Private** ou en **Static**, lorsqu'elle est **Private**, elle n'est utilisable que dans le module qui la contient, si elle est déclarée en **Public**, elle est disponible pour tout votre projet. Si votre **Sub** est déclarée en **Static**, la valeur de chaque variable déclarée dans votre procédure sera gardée entre deux appels (cette instruction n'a pas d'effets sur les variables déclarées hors de la procédure).

Ne rien spécifier équivaut à la déclarer en Public

Sans spécification

```
Sub MaProcedure()
```

Sans spécification

```
...
...
End Sub
```

Public

```
Public Sub MaProcedure()
...
...
End Sub
```

Private

```
Private Sub MaProcedure()
...
...
End Sub
```

Static

```
Static Sub MaProcedure()
...
...
End Sub
```

Essayons, créez un nouveau document, ouvrez le VBE. Ajoutez au document un nouveau module. Dans ce module, ajoutez ces quelques lignes :

```
Sub MaProcedure()
    MsgBox "Vient de ma procédure !"
End Sub
```

Dans le module de classe **ThisDocument** ajoutez ces quelques lignes :

```
Sub AppelProcedure()
    MaProcedure
End Sub
```

Placez le curseur de la souris dans le code et pressez F5, normalement, vous devriez avoir une boîte de message contenant le texte *Vient de ma procédure !*. Ce qui nous permet de dire que la procédure est utilisable par les autres procédures appartenant à d'autres modules de votre projet.

Ajoutons le mot réservé **Private** devant la procédure.

```
Private Sub MaProcedure()
    MsgBox "Vient de ma procedure !"
End Sub
```

Lançons à nouveau notre procédure

```
Sub AppelProcedure()
    MaProcedure
End Sub
```

Vous avez droit à un message d'erreur : **Erreur de compilation: Sub ou Fonction non définie**.

Copions notre procédure **Private** dans le module **ThisDocument** et recommençons l'opération. Tout se passe normalement nous avons à l'écran notre message : *Vient de ma procédure !*.

Nous connaissons maintenant quelle est la portée d'une procédure.

8-B - Les Function

Les **Function** sont similaires aux **Sub** avec une différence de taille, non pas sur le code contenu, mais sur le résultat renvoyé. Une **Function** renvoie un résultat alors qu'une **Sub** non.

Pour imager ce principe de résultat, nous allons utiliser une fonction VBA assez simple : **Len()**, cette fonction compte le nombre de caractères contenus dans une chaîne.

```
Function LongueurDuMot() As Integer
LongueurDuMot = Len("LongueurDuMot")
End Function
```

Si vous utilisez cette fonction, la valeur retournée de *LongueurDuMot* est un entier qui vaut **13**. Nous avons donc une valeur de type entier attribuée à *LongueurDuMot* (**LongueurDuMot() As Integer**). Nous verrons un peu loin dans l'appel des procédures ce que signifie exactement renvoyer un résultat.

8-B-1 - Portée d'une Function

Les Sub et Function étant similaires, leur portée est identique.

8-B-2 - Valeur renvoyée par une fonction

Si nous reprenons l'exemple précédent, notre fonction "Function LongueurDuMot() As Integer" ne renvoie qu'une seule valeur de type entier. Il arrive que nous ayons besoin de plusieurs valeurs. Si nous nous limitons à un type de données simple, nous allons avoir besoin d'autant de fonctions que de valeurs retournées.
Heureusement, il existe au moins une parade à cette limitation.

Si vous utilisez un type de données utilisateur, vous pouvez obtenir plusieurs résultats à votre fonction.

Plusieurs résultats

```
'**********************************************************
Public Type MyVar
    IntA As Integer
    IntB As Integer
End Type
'**********************************************************
Sub MaProc()
'Déclaration des variables
Dim MyData01 As MyVar
Dim MyData02 As MyVar
'Affectation des données
With MyData01
    .IntA = 10
    .IntB = 20
End With
MyData02 = monCalcul(MyData01)
Debug.Print MyData02.IntA, MyData02.IntB

End Sub
'**********************************************************
Public Function monCalcul(MyDt As MyVar) As MyVar
```

Plusieurs résultats
```
monCalcul.IntA = MyDt.IntA * 2
monCalcul.IntB = MyDt.IntB / 4

End Function
```

Si nous détaillons le code ci-dessus.

Nous avons une déclaration de type de données utilisateur composé de deux éléments de type entier.
Dans notre Sub, nous allons utiliser deux variables de type utilisateur, l'une qui va recevoir les données avant traitement et la seconde qui recevra les données après traitement (MyData01; MyData02).

La fonction reçoit en argument le type de données utilisateur et renvoie un type de données utilisateur. Sachant que ce type de données est composé d'éléments, nous avons bien en retour des valeurs multiples.

8-C - Appel d'une procédure

Nous avons vu dans la portée d'un Sub l'appel d'une autre routine.

```
Sub AppelProcedure()
    MaProcedure
End Sub
```

Certains programmeurs ont pour habitude de faire précéder l'appel d'une fonction ou d'une procédure par le mot **Call**. Ce mot est pour reprendre le terme utilisé par **Microsoft** facultatif. Dans l'exemple, que vous utilisiez Call ou non ne change rien.

```
Sub AppelProcedure()
    MaProcedure
End Sub
```

Est identique à

```
Sub AppelProcedure()
    Call MaProcedure
End Sub
```

Lorsque vous faites appel à une procédure, vous pouvez lui passer des paramètres qui seront alors utilisés.

Reprenons notre exemple de départ.

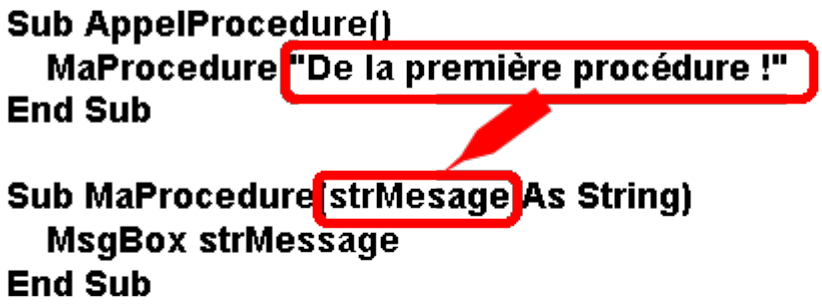
```
Sub MaProcedure()
    MsgBox "Vient de ma procedure !"
End Sub


Sub AppelProcedure()
    MaProcedure
End Sub
```

Nous pourrions passer le texte de notre boîte de message à notre procédure.
Essayons :

```
Sub AppelProcedure()
```

```
    MaProcedure "De la première procédure !"
End Sub

Sub MaProcedure(strMessage As String)
    MsgBox strMessage
End Sub
```

Le texte affiché dans la boîte de dialogue correspond au texte contenu dans la première procédure.



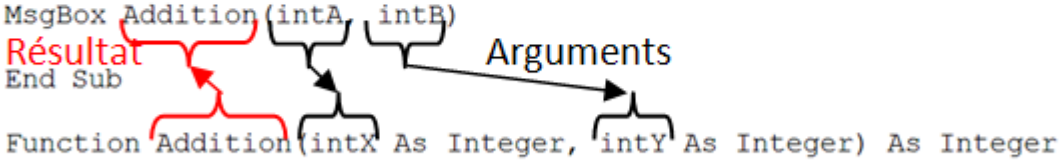
Nous passons donc le texte ***"De la première procédure !"*** en argument pour l'appel de la seconde procédure et c'est ce texte qui est utilisé pour l'affichage de notre boîte de dialogue.

Nous allons faire un exercice avec une Function, lui passer des valeurs en arguments et récupérer le résultat. Nous allons utiliser une simple addition de nombres.

```
Sub AppelFonction()
'Déclaration des variables
Dim intA As Integer
Dim intB As Integer
Dim strQuestion As String
'Affectation des variables
strQuestion = "Entrez un nombre entier."
intA = InputBox( strQuestion)
intB = InputBox( strQuestion)

MsgBox Addition(intA, IntB)

End Sub

Function Addition( intX As Integer, intY As Integer) As Integer
    Addition = intX + intY
End Function
```

Le principe est le suivant, lors de l'appel, on passe deux nombres en argument et la fonction retourne un résultat.

```
Sub AppelFonction()
'Déclaration des variables
Dim intA As Integer
Dim intB As Integer
Dim strQuestion As String
'Affectation des variables
strQuestion = "Entrez un nombre entier."
intA = InputBox(strQuestion)
intB = InputBox(strQuestion)

MsgBox Addition(intA, intB)
End Sub

Function Addition(intX As Integer, intY As Integer) As Integer
    Addition = intX + intY
End Function
```



Si tout s'est déroulé correctement, vous avez reçu une boîte de message contenant la somme des deux nombres préalablements introduits.

8-C-1 - ByVal ou ByRef

Lorsque nous passons des arguments à une fonction, nous avons deux méthodes pour les passer, ByVal ou ByRef. Dans le premier cas, nous passons la valeur de la variable, nous pourrions comparer ça au passage d'une copie de la valeur à la fonction. Tout traitement effectué sur la valeur reçue n'aura pas d'influence sur la valeur de départ.
Si nous passons la valeur ByRef, c'est la référence à la valeur qui est passée et toute modification de cette valeur "reçue" se répercutera sur la valeur de départ puisque c'est la même.

Nous allons utiliser un exercice pour mieux comprendre.

Par Valeur

```vba
'Déclaration d'une variable de type entier
Public intA As Integer

Sub TestOli()
'Affectation d'une valeur à la variable
intA = 10
'Deux MsgBox de Contrôle
MsgBox TestModInt(intA)
MsgBox "Valeur de intA : " & intA
End Sub

Function TestModInt(ByVal intB As Integer) As Integer
intB = intB / 2
TestModInt = 1
End Function
```

La valeur affichée dans notre MsgBox est 10, elle n'est donc pas modifiée.

Effectuons le même test mais en passant l'agument ByRef.

Par Référence

```
'Déclaration d'une variable de type entier
Public intA As Integer

Sub TestOli()
'Affectation d'une valeur à la variable
intA = 10
'Deux MsgBox de Contrôle
MsgBox TestModInt(intA)
MsgBox "Valeur de intA : " & intA
End Sub

Function TestModInt(ByRef intB As Integer) As Integer
intB = intB / 2
TestModInt = 1
End Function
```

Nous recevons deux MsgBox, la première affichant 1 et la seconde 5. Ce qui nous prouve que la valeur d'origine de notre variable a été modifiée.

*Si vous ne spécifiez pas **ByRef** ou **ByVal**, c'est ByRef qui est utilisé par défaut.*

8-C-2 - Optional

Si certaines valeurs peuvent être manquantes dans les arguments, vous devez les déclarer avec l'attribut **Optionnal** pour ne pas lever d'erreur.

```
Public Fonction MaSuperFonction( intA As Integer, intB As Integer, Optional IntC As Integer, Optional
 intD As Integer) As String
....
End Function
```

Dans la déclaration de vos arguments, vous ne pouvez pas mélanger les facultatifs et les obligatoires. Vous devez commencer par les obligatoires et finir par les facultatifs.

```
Public Fonction MaSuperFonction( intA As Integer, Optional  intB As Integer, IntC As Integer, Optional
 intD As Integer) As String
....
End Function
```

Cette dernière est fautive et générera une erreur de compilation.

9 - Écriture de votre code

Lorsque vous allez commencer à écrire vos premières lignes de code, vous devrez tenir compte de certaines petites choses.

9-A - Les commentaires

En VBA, les commentaires sont toujours précédés d'une apostrophe ('). Dès que l'éditeur rencontre ce caractère, il colore en Vert (couleur par défaut) le texte qui suit. Lors de la compilation de votre code, ce texte est ignoré. Mais ô combien utile lorsque vous devez vous relire ou passer votre travail à quelqu'un d'autre.

```
'Déclaration d'une variable de type entier
```

```
Public intA As Integer
```

Les commentaires peuvent se trouver n'importe où, mais placez les à des endroits judicieux.

9-B - Les retours à la ligne

Malheureusement, la taille des écrans est souvent beaucoup plus petite que ce dont nous avons besoin, surtout en largeur. Il est possible en VBA d'écrire une ligne de code sur plusieurs lignes consécutives. Pour y parvenir, il suffit d'utiliser un espace suivi d'un caractère de soulignement et la ligne de code se poursuit sur la ligne suivante.

```
Public Fonction MaSuperFonction( intA As Integer, intB As Integer, Optional IntC As Integer, Optional
 intD As Integer) As String
....
End Function
```

Peut aussi s'écrire :

```
Public Fonction MaSuperFonction( intA As Integer, intB As Integer, _
     Optional IntC As Integer, Optional intD As Integer) As String
....
End Function
```

9-C - L'indentation

En plus des commentaires et de l'écriture des lignes de code sur plusieurs lignes, il existe une règle simple pour faciliter la compréhension et la lecture mais pas obligatoire, l'indentation du code.

```
Sub TestOli()
'Affectation d'une valeur à la variable
intA = 10
'Deux MsgBox de Contrôle
MsgBox TestModInt(intA)
MsgBox "Valeur de intA : " & intA
End Sub
```

Dans le code qui précède, le nombre de lignes est très faible et l'indentation n'est pas nécessaire, mais pour servir d'exemple, nous allons indenter les lignes de code.

```
Sub TestOli()
'Affectation d'une valeur à la variable
 intA = 10
'Deux MsgBox de Contrôle
 MsgBox TestModInt(intA)
 MsgBox "Valeur de intA : " & intA
End Sub
```

C'est un peu plus lisible. Prenez l'habitude de systématiquement indenter votre code.

10 - Les opérateurs

10-A - Affectation (=)

Nous avons déjà utilisé cet opérateur dans les articles précédents.

10-B - Arithmétiques

Il existe en VBA des opérateurs arithmétiques pour manipuler les données numériques. Le tableau ci-dessous reprend la liste des opérateurs disponibles.

Opérateur	Utilisation	Exemple
+	Additionner deux nombres	Dim intI As Integer, intJ As integer Dim intK As Double intI = 14 IntJ = 3 IntK = intI + intJ intK = 17
-	Soustraire deux nombres	intK = intI □ intJ intK = 11
*	Multiplier deux nombres	intK = intI * intJ intK = 42
/	Diviser deux nombres	intK = intI / intJ intK = 4,6666667
^	Exposant	intK = intI ^ intJ intK = 2744
Mod	Modulo, renvoie le reste de la division	intK = intI mod intJ intK = 2
\	Renvoie la partie entière de la division	intK = intI \ intJ intK = 4

10-C - Comparaison

En plus des opérations arithmétiques, vous serez amenés à faire des comparaisons de données, le tableau ci-dessous vous donne les principaux opérateurs de comparaison.

Opérateur	Utilisation
A<B	Plus petit
A<=B	Plus petit ou égal
A>B	Plus grand
A=>B	Plus grand ou égal
A=B	Égal
A<>B	Différent

Il existe deux opérateurs de comparaison supplémentaires qui sont Is et Like. Le premier est applicable aux objets et le second pour les comparaisons de chaînes de caractères avec ou sans WildCard.

10-D - Concaténation

Il existe deux opérateurs de concaténation l'esperluette "&" et le plus "+".
Il est préférable d'utiliser le &, si vous concaténez avec l'opérateur + deux valeurs où l'une d'entre elles est Null, le résultat sera Null, alors que si vous utiliser le &, vous obtiendrez le résultat de la valeur non Null. Si vous avez d'autres types de données que des strings, l'opérateur + va faire une simple addition.

10-E - Opérateurs logiques

Finalement, nous allons traiter les principaux opérateurs logiques. Nous allons les découvrir un par un et sous forme de tableau pour plus de facilité. Nous ne traiterons que les résultats obtenus avec des booléens.

La structure est toujours la même, **Expression** *Opérateur* **Expression** => **Résultat**.

10-E-1 - Or

Le traduction est OU, si l'une des deux valeurs est vraie, le résultat est vrai.

Expression 1	Expression 2	Résultat
1	1	1
1	0	1
0	1	1
0	0	0

10-E-2 - And

La traduction est ET, il faut que les deux expressions soient vraies pour que le résultat le soit.

Expression 1	Expression 2	Résultat
1	1	1
1	0	0
0	1	0
0	0	0

10-E-3 - XOR

Si une seule des expressions a pour valeur Vrai, l'argument result est Vrai. C'est un OU exclusif.

Expression 1	Expression 2	Résultat
1	1	0
1	0	1
0	1	1
0	0	0

10-E-4 - NOT

Permet d'établir la négation logique d'une expression.

Expression	Résultat
1	0
0	1

Cette liste n'est certainement pas exhaustive.

11 - Les structures décisionnelles

Après les opérateurs, nous allons aborder les structures décisionnelles.
La structure décisionnelle permet à une routine de faire une action ou une autre en fonction de certain paramètre.

Si nous devions placer ce genre de structure dans la vie de tous les jours, nous pourrions écrire ceci :
Quel temps fait-il ?
S'il pleut nous prenons le parapluie.
S'il fait sec, nous ne prenons pas le parapluie.

En structure informatique nous aurions ceci :

```
Si Pluie = Vrai Alors
    Prendre parapluie
Sinon
    Pas prendre parapluie
...
```

11-A - La structure If ... Then ...

C'est la plus simple des structures décisionnelles. On pose une condition, si elle est remplie, on effectue une action.

```
If Condition Then Instruction
```

En code utilisable, nous obtenons ceci :

```
Sub Vetements()
Dim dblTemperature As Double
dblTemperature = InputBox("Quelle température fait-il ?")
If dblTemperature < 5 Then MsgBox "Prends une écharpe, il fait froid !"
End Sub
```

Dans cette structure, pour les températures supérieures à 5°, il ne se produit rien.

11-B - La structure If ... Then ... ElseIf ... Else... End If

La structure If Then peut être complétée par des clauses supplémentaires et cette structure sera écrite sur plusieurs lignes.

```
If Condition Then
 Instruction si Vrai
Else
 Instruction si Faux
End If
```

En pratique on obtient ceci :

```
Sub Vetements()
' Déclaration des variables
Dim dblTemperature As Double
dblTemperature = InputBox("Quelle température fait-il ?")
'Structure décisionnelle
 If dblTemperature < 5 Then
  MsgBox "Prends une écharpe, il fait froid !"
 Else
  MsgBox "Pas besoin d'écharpe aujourd'hui !"
 End If
End Sub
```

Dans cette structure, nous avons un résultat supplémentaire, si la température est supérieure à 5°, nous n'avons pas besoin d'écharpe.

Cette structure est extensible, on peut lui ajouter d'autres clauses.

```
Sub Vetements()
' Déclaration des variables
Dim dblTemperature As Double
dblTemperature = InputBox("Quelle température fait-il ?")
'Structure décisionnelle
 If dblTemperature < 5 Then
  MsgBox "Prends une écharpe, il fait froid !"
 ElseIf dblTemperature > 20 Then
  MsgBox "Un T-shirt sera le bon choix !"
 Else
  MsgBox "Pas besoin d'écharpe aujourd'hui !"
 End If
End Sub
```

Ce code possède trois sorties, si la température est inférieure à 5°, si la température est comprise entre 5° et 20° et finalement si la température est supérieure à 20°. Ajouter une ou plusieurs clauses supplémentaires rendrait le code trop complexe.

11-C - La structure Select Case

La structure Select Case est très intuitive. Elle exécute un des blocs d'instructions indiqués, selon la valeur d'une expression. Un peu comme lorsque l'on cherche un bureau dans un couloir, on regarde ce qui est écrit sur la porte et si c'est la bonne porte nous entrons.
Nous allons reprendre le dernier exemple utilisé.

```
Sub Vetements()
' Déclaration des variables
Dim dblTemperature As Double
dblTemperature = InputBox("Quelle température fait-il ?")
'Structure décisionnelle
 Select Case dblTemperature 'Expression qui sera testée
  Case Is < 5
   MsgBox "Prends une écharpe, il fait froid !"
  Case 5 To 20
   MsgBox "Pas besoin d'écharpe aujourd'hui !"
  Case Is > 20
   MsgBox "Un T-shirt sera le bon choix !"
  End Select
End Sub
```

Tout au long de la structure de test, on va comparer la valeur contenue dans notre expression aux valeurs de test. Si nous désirons ajouter des conditions supplémentaires, c'est assez simple.

12 - Les boucles

Les boucles permettent de répéter un certain nombre de fois une même action. Il existe plusieurs syntaxes pour les boucles, on ne peut pas dire que l'une soit meilleure que l'autre. C'est à vous de décider quelle sera la meilleure boucle à mettre en oeuvre dans votre code.
Le principe des boucles reste toujours le même, on pose une condition et tant que cette condition n'est pas remplie, on exécute le contenu de la boucle.
Dans la vie de tous les jours, les exemples sont nombreux, remplir un verre d'eau en fait partie, tant que le verre n'est pas rempli, je verse, lorsqu'il est rempli, j'arrête et je le bois.

Si vous utilisez des boucles, vous devez vous assurer que le code va bien sortir de la boucle.

Imaginez une boucle où vous stipulez : tant que X n'est pas égal à 10, on ajoute 1. Si vous commencez la boucle à 0, elle passera par 10 et donc s'arrêtera, si vous commencez à 11, vous entrez dans une boucle infinie.

Si vous entrez dans une boucle infinie, l'utilisation de Ctrl + Break ou Ctrl + Pause va arrêter le déroulement de votre code.

Avant de lancer une boucle, n'oubliez pas d'initialiser ou réinitialiser vos variables.

Les boucles peuvent être imbriquées les unes dans les autres.

12-A - Do While .... Loop

Tant que la condition qui suit While est vraie, la boucle est exécutée, si la condition devient fausse, les instructions se trouvant dans la boucle ne sont plus exécutées et le code continue son déroulement.

Do While ... Loop

```
Sub TestBoucle()
Dim intI As Integer
'initialisation de la variable
intI = 0
Do While intI < 10
    intI = intI + 1
    MsgBox intI
Loop
End Sub
```

La syntaxe de l'expression testée est très importante. Dans l'exemple donné, la boucle va s'arrêter sur intI = 10, si vous utilisez <= dans votre expression, la valeur de intI sera de 11, puisque la boucle vérifie la condition en début de cycle, et lorsque intI sera égal à 10, la boucle sera une dernière fois exécutée.

On pourrait traduire cette boucle par : Tant que

12-B - While .... Wend

Cette boucle est identique à la précédente, seule la manière de l'écrire change.

While ... Wend

```
Sub TestBoucle()
Dim intI As Integer
'initialisation de la variable
```

While ... Wend
```
intI = 0
While intI < 10
    intI = intI + 1
    MsgBox intI
Wend
End Sub
```

12-C - Do Until .... Loop

Cette boucle est juste le contraire de la précédente, elle est exécutée tant que la condition qui suit Until est fausse, si cette condition devient vraie, les instructions de la boucle ne sont pas exécutées.

Do Until ... Loop
```
Sub TestBoucle()
Dim intI As Integer
'initialisation de la variable
intI = 0
Do Until intI > 10
    intI = intI + 1
    MsgBox intI
Loop
End Sub
```

Si vous essayez ce code, vous vous apercevrez que le résultat est sensiblement le même que pour la boucle précédente, seule la condition change.

On pourrait traduire cette boucle par : Jusqu'à ce que

12-D - Do .... Loop While

À l'inverse des boucles précédentes où la condition est vérifiée à l'entrée de la boucle, dans les deux boucles suivantes, la condition est vérifiée à la sortie de la boucle.

Do ... Loop While
```
Sub TestBoucle()
Dim intI As Integer
'initialisation de la variable
intI = 0
Do
    intI = intI + 1
    MsgBox intI
Loop While intI < 10
End Sub
```

Tant que la condition est vraie, la boucle est effectuée.

12-E - Do ... Loop Until

Comme pour la boucle précédente, la condition est vérifiée à la sortie de la boucle, mais la boucle est effectuée tant que la condition de test n'est pas vérifiée.

Do ... Loop Until
```
Sub TestBoucle()
Dim intI As Integer
'initialisation de la variable
intI = 0
```

Do ... Loop Until
```
Do
    intI = intI + 1
    MsgBox intI
Loop Until intI > 10
End Sub
```

Jusqu'à ce que la condition soit vraie, la boucle est effectuée.

*Il est possible de sortir de la boucle avec une instruction **Exit Do**.*
*Cette instruction est valable pour toutes les boucles **Do***

```
If Condition Then Exit Do
```

12-F - For ... To .... Next

Cette boucle est basée sur un "compteur" numérique. La valeur d'incrémentation de ce compteur est donnée par l'instruction **Step**. Si cette instruction est omise, la valeur de Step est 1. Cette valeur peut être négative, dans ce cas, nous obtenons une décrémentation.

For ... To ... Next
```
Sub TestBoucle()
Dim intI As Integer
'initialisation de la variable
intI = 0
For intI = 0 To 10 Step 1
 MsgBox intI
Next intI
End Sub
```

Avec cette boucle, tous les nombres entiers entre 0 et 10 sont affichés. Tentons la même expérience avec un Step de 2.

Avec un step de 2
```
Sub TestBoucle()
Dim intI As Integer
'initialisation de la variable
intI = 0
For intI = 0 To 10 Step 2
 MsgBox intI
Next intI
End Sub
```

Nous n'obtenons que les nombres pairs de la suite.

12-G - For Each .... Next ...

Cette boucle est utilisée pour parcourir les éléments d'une collection, liste, tableau,...
L'exemple donné ne fonctionne pas, nous aurons l'occasion d'utiliser cette boucle dans les tutoriels suivants.

For Each ... Next
```
Sub TestBoucle()
Dim objElem As Element
For Each objElem In Collection.Elements
 ....
Next objElem
```

For Each ... Next
```
End Sub
```

Il est possible de sortir de la boucle avant que la condition finale ne soit atteinte. Cette instruction est valable pour toutes les boucles **For**.

Sortie de la boucle For
```
If Condition Then Exit For
```

13 - Quelques fonctions VBA

Les fonctions qui vont suivre sont applicables à tous les logiciels qui sont capables d'utiliser le VBA. Ce sont des fonctions générales.
Ces fonctions peuvent être appliquées aux données ou aux variables, dans ce dernier cas, c'est le contenu de la variable qui sera affecté par la fonction et pas la variable.

13-A - Fonctions Texte

13-A-1 - Len

Cette fonction renvoie la longueur d'une chaîne de caractères passée en argument.

La fonction Len
```
Sub LongueurChaine()
 MsgBox Len("www.Developpez.Com")
End Sub
```

Vous devriez avoir une boîte de message avec 18.
Vous pouvez aussi utiliser les fonctions sur les variables et c'est le contenu de la variable qui est utilisé.

Len sur une variable
```
Sub LongueurChaine()
 Dim strMaVar As String
 strMaVar = "www.Developpez.Com"
 MsgBox Len(strMaVar)
End Sub
```

Le résultat est identique.

13-A-2 - UCase

Cette fonction va basculer une chaîne de caractères en Majuscules, le pendant de cette fonction est **LCase**.

UCase
```
Sub MiseEnMajuscules()
 MsgBox UCase("www.developpez.com")
End Sub
```

Le résultat : WWW.DEVELOPPEZ.COM

Les fonction UCase et LCase sont très importantes lors d'une recherche de chaîne. Soit vous basculez les deux en majuscules, soit vous basculez les deux en minuscules. Dans un cas comme dans l'autre, vous éviterez le problème lié au mélange des deux.

13-A-3 - LCase

Si UCase bascule une chaîne de caractères en majuscules, LCase bascule une chaîne en minuscules.

LCase

```
Sub MiseEnMinuscules()
 MsgBox LCase("www.Developpez.Com")
End Sub
```

Donnera comme résultat www.developpez.com

13-A-4 - Left

Cette fonction va renvoyer la partie gauche d'une chaîne de caractères constituée du nombre de caractères passé en argument.

Left

```
Sub FonctionLeft()
 MsgBox Left("www.developpez.com", 3)
End Sub
```

Le résultat est "www"

13-A-5 - Right

Si la fonction Left renvoie la partie gauche d'une chaîne de caractère, la fonction Right renvoie la partie droite d'une chaîne.

Right

```
Sub FonctionRight()
 MsgBox Right("www.developpez.com", 3)
End sub
```

Le résultat est "com"

13-A-6 - Mid

La fonction Mid est à mi chemin entre la fonction Left et la fonction Right, elle renvoie un morceau de la chaîne qui va commencer au caractère représenté par le second argument et du nombre de caractère représenté par le dernier argument.

Mid

```
Sub FonctionMid()
 MsgBox Mid("www.developpez.com", 4, 12)
End Sub
```

Le résultat est ".developpez."

13-A-7 - Chr

La fonction Chr renvoie le code une lettre au départ de son code ASCII. Cette fonction est très intéressante lorsque vous devez utiliser des caractères non autorisés dans des expressions.

Chr

```
Sub FonctionChr()
 MsgBox Chr(65)
End Sub
```

Le résultat est A

Les deux caractères qui sont le plus utilisés sont le 13 et le 34, je vous laisse essayer !

13-A-8 - Asc

Si Chr renvoie une lettre en fonction de son code ASCII, la fonction **Asc** renvoie le code ASCII d'une lettre.

```
Sub FonctionAsc()
 MsgBox Asc("A")
End sub
```

Le résultat est 65

Passons à un exercice un peu plus complexe. Nous allons utiliser quelques fonctions précédemment vues pour obtenir le code ASCII d'une chaîne de caractère.
Nous allons également mettre en pratique les boucles. La chaîne que nous allons utilisez est : www.developpez.com
Essayez de résoudre cet exercice, pensez aux méthodes que vous allez utiliser et comment vous allez les mettre en oeuvre.

Les fonctions à utiliser sont Len() et Mid().

Décomposer une chaîne

```
Sub DecomposerChaine()
'Déclaration des variables
Dim strMaCh As String
Dim intI As Integer
'Affectation des données aux variables
strMaCh = "www.developpez.com"
'une boucle qui va compter de la première à la dernière lettre de notre variable
For intI = 1 To Len(strMaCh)
 'Pour ne pas devoir cliquer sur OK pour toutes les lettres, j'ai préféré l'utilisation
 'du Debug.Print, le résultat apparaît dans la fenêtre exécution (Ctrl + G)
 'La première partie affiche la lettre et la seconde son code
    Debug.Print Mid(strMaCh, intI, 1) & " - " & Asc(Mid(strMaCh, intI, 1))
Next intI

End Sub
```

Petites explications :
La boucle va parcourir avec un pas de 1 toutes les lettres de l'expression, de 1 à **Len(strMaCh)**.
Pour chaque valeur de intl, nous allons récupérer la lettre correspondante, **Mid(strMaCh, intl, 1)**, nous prenons une lettre et cette lettre se situe à intl du début.
Pour cette même lettre, nous récupérons sa valeur ASCII avec **Asc(Mid(strMaCh, intl, 1))**.

Le tout avec un Debug.Print pour l'afficher dans la fenêtre exécution, nous aurions pu utiliser un MsgBox, mais c'est moins convivial.

Le résultat :

w - 119
w - 119
w - 119
. - 46
d - 100
e - 101
v - 118
e - 101
l - 108
o - 111
p - 112
p - 112
e - 101
z - 122
. - 46
c - 99
o - 111
m - 109

13-A-9 - Trim - LTrim - RTrim

Les fonctions Trim suppriment les espaces d'une chaîne de caractères.
LTrim supprime les espaces situés à gauche de la chaîne.
RTrim supprime les espaces situes à droite de la chaîne.
Trim supprime les espaces à gauche et à droite.

Trim

```
Sub FonctionTrim()
 Debug.Print Trim ("    Bonjour Monsieur ")
End sub
```

Le résultat est "Bonjour Monsieur"

13-A-10 - StrReverse

Le nom de la fonction devrait suffire. Cette fonction inverse l'ordre des caractères d'une chaîne.

StrReverse

```
Sub DecomposerChaine()
Dim strMaCh As String

strMaCh = "www.developpez.com"

Debug.Print StrReverse(strMaCh)

End Sub
```

Le résultat : **moc.zeppoleved.www**

13-A-11 - Replace

Renvoie une chaîne dans laquelle une sous-chaîne spécifiée a été remplacée plusieurs fois par une autre sous-chaîne.

Replace

```
Sub FonctionReplace()
Dim strMyChar As String
Dim strMyRepl As String

strMyChar = "www.developpez.com"
strMyRepl = "heureuxoli"

Debug.Print "Avant remplacement : " & strMyChar
Debug.Print "Après remplacement : " & Replace(strMyChar, "www", strMyRepl)


End Sub
```

Le résultat obtenu
Avant remplacement : www.developpez.com
Après remplacement : heureuxoli.developpez.com

13-A-12 - InStr

Cette fonction permet de récupérer la position d'un caractère dans une chaîne de caractères. Seule la première occurrence est renvoyée.

InStr

```
Sub FonctionInStr()
Dim strMyChar As String

strMyChar = "www.developpez.com"

Debug.Print InStr(1, strMyChar, "d")

End Sub
```

Le résultat est 5

13-A-13 - Split

Cette fonction éclate une chaîne de caractères en un tableau en fonction d'un caractère spécifié. Mais un exemple sera plus concret.
Utiliser les variables tableaux en VBA

Split

```
Sub FonctionSplit()
'Déclaration des variables
Dim strMyChar As String
Dim strMyResult() As String
Dim inti As Integer
'Affectation des données aux variables
strMyChar = "www.developpez.com"
'Utilisation de la fonction
strMyResult = Split(strMyChar, ".")
'Boucle pour l'affichage du contenu de la table
For inti = 0 To 2
Debug.Print inti & " - " & strMyResult(inti)
Next inti

End Sub
```

Nous connaissons avant de commencer le nombres d'éléments qui vont constituer notre tableau, 3. Le tableau renvoyé a comme plus petit indice 0. L'utilisation de la fonction Split va affecter trois valeurs à notre tableau, "**www**" dans strMyResult(0); "**developpez**" dans strMyResult(1) et "**com**" dans strMyResult(2). Nous avons donc bien éclaté notre chaîne de départ contenant deux points en trois parties.

13-B - Fonctions de conversion

Pour les fonctions de conversion, elles vont être données sous la forme d'un tableau et ne seront pas détaillées.

Fonction	Usage	Exemple	Résutat
CBool	Convertir une chaîne en booléen	CBool("0")	0
CByte	Convertir une expression en donnée de type Byte	CByte("124")	124
CCur	Convertir une expression en valeur monétaire	CCur(125,364)	125.364
CDate	Convertir une expression en date	CDate("01/05/2008")	#01/05/2008#
CDbl	Convertir une expression en double	CDbl(#01/05/2008#)	39580
CDec	Convertir une expression en décimal	CDec("0,25")	0.25
CInt	Convertir une expression en Entier	CInt("125")	125
CLng	Convertir une expression en Long	CLng (125)	125
CSng	Convertir une expression en simple	CSng("12")	12
CStr	Convertir un nombre en chaîne	CStr(12)	"12"
CVar	Convertir une expression en Variant	CVar(12)	12

13-B-1 - Str

Fonction permettant de convertir une valeur numérique en texte.

Str

```
Str(123) 'Renvoie " 123"
```

13-C - Fonctions Date

Si vous désirez avoir une description plus complète que celle de ce tutoriel, je vous conseille de lire cet **article**.

13-C-1 - Date()

Cette fonction renvoie la date système.

Date

```
Sub FonctionDate()
Debug.Print Date()
End Sub
```

13-C-2 - Now()

Cette fonction renvoie la date et l'heure système.

Now

```
Sub FonctionNow()
Debug.Print Now()
End Sub
```

13-C-3 - DateAdd

Cette fonction permet d'ajouter une valeur à une date, cette valeur peut être en secondes, minutes, heures, jours, mois ou années.

```
Sub FonctionDateAdd()
Debug.Print DateAdd("d", 5, #27/02/2008#)
End Sub
```

Le résultat est #03/03/2008#, le 29/02/2008 a donc bien été pris en compte.

Interval	Description
yyyy	Année
m	Mois
d	Jour
h	Heure
n	Minute
s	Seconde

13-C-4 - DateDiff

Alors que la fonction DateAdd permet d'ajouter un intervalle à une date, DateDiff permet au départ de deux dates de retrouver l'intervalle qui les sépare.

DateDiff

```
Sub FonctionDateDiff()

Debug.Print DateDiff("d", #19/10/1964#, Date)

End Sub
```

Essayez de remplacer le 19/10/1964 par votre date de naissance, vous connaîtrez le nombre de jours écoulé depuis votre naissance.

Interval	Description
yyyy	Année
m	Mois
d	Jour
h	Heure
n	Minute
s	Seconde

13-C-5 - DatePart

Cette fonction renvoie une partie d'une date en fonction de l'intervalle passé en argument.

DatePart

```
Sub FonctionDatePart()
Debug.Print DatePart("m", Date)

End Sub
```

Interval	Description
yyyy	Année
q	Trimestre
m	Mois
d	Jour
h	Heure
n	Minute
s	Seconde

13-C-6 - DateSerial

Cette fonction renvoie une date en fonction d'un jour, d'un mois et d'une année.

DateSerial

```
Sub FnctionDateSerial()
Debug.Print DateSerial(2008, 12, 15)
End Sub
```

Cette fonction peut avoir plusieurs avantages, elle est très pratique pour renvoyer le premier jour du mois suivant.

Premier Jour Mois Suivant

```
Sub FonctionPremierJourMoisSuivant()
Debug.Print DateSerial(Year(Date), Month(Date) + 1, 1)
End Sub
```

Si on décortique ce code, on utilise la fonction **Year()** qui renvoie l'année d'une date, **Month()** qui renvoie le mois d'une date et finalement 1 pour le jour du mois. La date que nous utilisons est la date système renvoyée par la fonction **Date()** et pour le mois, nous utilisons également la date système. Pour obtenir le mois suivant, nous ajoutons 1 au mois.

13-C-7 - DateValue

Renvoie une date au départ d'une chaîne de caractère.

DateValue

```
Sub FonctionDateValue()
Debug.Print DateValue("15 décembre 2008")
End Sub
```

Le résultat est #15/12/2008#

13-D - Fonctions Système

13-D-1 - Beep

La fonction Beep est la fonction la plus énervante que je connaisse.

```
Sub FonctionBeep()
Dim intI As Integer
Dim intJ As Long
For intI = 1 To 10
  Beep
```

```
    intJ = 0
     For intJ = 1 To 10000000
    Next intJ
Next intI
End Sub
```

13-D-2 - Environ

Cette fonction permet de renvoyer certaines informations système. La plus "célèbre" est le UserName qui renvoie le nom de l'utilisateur courant.

```
Sub FonctionEnviron()
Debug.Print Environ("UserName")
End Sub
```

*Si vous voulez connaître les informations que environ peut renvoyer, allez en ligne de commande et tapez **SET**.*

13-D-3 - DoEvents

Arrête momentanément l'exécution afin que le système d'exploitation puisse traiter d'autres événements.

DoEvents

```
DoEvents
```

13-D-4 - CreateObject

Cette fonction permet de créer un objet et renvoie la référence de cet objet.

CreateObject

```
Sub FonctionCreateObject()
Dim objMyObject As Object

Set objMyObject = CreateObject("Excel.Application")

objMyObject.Quit
Set objMyObject = Nothing
End Sub
```

Chaque fois que vous faites appel à cette fonction, une nouvelle instance de l'objet est créée.

N'oubliez pas de libérer les objets quand vous n'en avez plus besoin.

13-D-5 - GetObject

Alors que la fonction CreateObject crée une instance, la fonction GetObject va faire référence à une instance existante de l'objet. Si aucune instance n'est ouverte, elle renvoie une erreur.

```
Sub FonctionGetObject()
Dim objMyObject As Object

Set objMyObject = GetObject("Excel.Application")
```

```
Set objMyObject = Nothing
End Sub
```

13-E - InputBox - MsgBox

Vous avez déjà fait connaissance avec ces fonctions dans les tutoriels précédents. Pour la fonction **InputBox**, le résultat doit être affecté à une variable ou à une autre fonction.
La fonction MsgBox peut quant à elle être utilisée seule pour avertir l'utilisateur, mais elle peut aussi servir à recueillir une information par le biais d'un bouton de réponse.

InputBox

```
Sub FonctionInputBox()
Dim strMyVar As String

strMyVar = InputBox("Entrez un texte !", "Votre Texte")
MsgBox "Votre texte comporte " & Len(strMyVar) & " caractères"
End Sub
```

Lorsque vous utilisez la fonction MsgBox pour obtenir des informations, vous devez affecter son résultat à une variable ou à une fonction.

MsgBox

```
Sub FonctionMsgBox()
Dim strMyResponse As Integer

strMyResponse = MsgBox("Êtes vous d'accord ?", vbOKCancel, "Votre choix")
If strMyResponse = 1 Then
    MsgBox "Vous avez cliqué sur OK !", , "OK"
Else
    MsgBox "Vous avez cliqué sur Annuler !", , "Annuler"
End If
End Sub
```

Voilà un exemple avec les deux possibilités d'utilisation d'un MsgBox.

Si vous voulez explorer les boîtes de message, un peu de lecture.

14 - Liens Utiles

	Titre de l'article
	Fondements sur les variables et les constantes
	Les conventions typographiques en VBA (illustrées sur Access)
	Initiation au VBA d'Outlook

15 - Remerciements

Je tiens à remercier pour leurs contributions à ce projet :
- **lorenzole+bo**
- **Philippe JOCHMANS**
- **Caro-Line**
- **Arkham46**
- **Jeannot45**