Les fonctions personnelles dans Excel

par SilkyRoad (silkyroad.developpez.com)

Date de publication : 26/11/2007

Dernière mise à jour :

Ce tutoriel présente la création et l'utilisation des **fonctions** complémentaires dans Excel.

Les exemples proposés ont été testés avec Excel 2002 et 2007.

I - Introduction

L'environnement Excel met à votre disposition un large panel de fonctions prédéfinies. Si vous ne trouvez pas votre bonheur dans les outils existants, une solution consiste à créer vos fonctions personnelles. C'est l'objet de cet article qui montre comment créer et utiliser des fonctions complémentaires. Ces dernières sont utilisées de la même manière que les fonctions de l'application.

Contrairement aux procédures **Sub** qui ne renvoient pas de données, une fonction (**Function**) est un type de procédure qui retourne une valeur.

Les fonctions commencent par l'instruction **Function** et se terminent par l'instruction **End Function**. Le code doit être écrit entre ces deux instructions. Vous pouvez définir le type de résultat retourné de la même manière que pour **une variable**.

Le code doit être placé dans un module standard si vous souhaitez utiliser les fonctions dans des formules de la feuille de calcul.

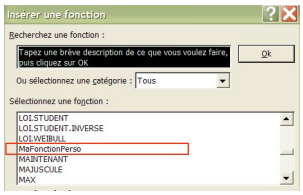
Les fonctions acceptent des arguments obligatoires et optionnels.

Remarque importante:

Les fonctions personnelles ne sont pas disponibles lorsque les macros sont désactivées. Dans ce cas, la formule renvoie la valeur d'erreur #NOM?.

Les fonctions n'apparaissent pas dans la boîte de dialogue des macros (Alt+F8). Par contre elles sont intégrées dans la liste des fonctions Excel tant que le classeur qui les contient est ouvert.

Pour la visualiser dans la liste, utilisez le menu Insertion/Fonction/Catégorie "Personnalisées" (ou onglet "Formules"/groupe "Bibliothèque de fonctions"/bouton "Insérer une fonction", dans Excel2007):



Si vous souhaitez intégrer une fonction personnelle dans des formules de la feuille de calcul, il est très important de noter que Les instructions permettent uniquement de renvoyer une donnée dans la cellule. Ces instructions ne peuvent pas modifier les différents objets du classeur.

II - Créer une fonction

II-A - Un exemple pour débuter

Tout d'abord, voici comment mettre en oeuvre votre première fonction.

Vous allez employer **l'éditeur de macros**.

Ouvrez un nouveau classeur vierge.

Pour atteindre l'éditeur, utilisez le menu Outils/Macro/Visual Basic Editor.

Si vous disposez d'Excel2007, sélectionnez l'onglet "Développeur" puis cliquez sur le bouton "Visual Basic" dans le groupe "Code".

Vous pouvez aussi ouvrir l'éditeur à partir du raccourci clavier **ALT+F11**.

Ajoutez un module standard en sélectionnant le menu Insertion/Module.

Copiez la procédure suivante dans le module. L'exemple multiplie par 2 la valeur transmise à la fonction.

Vba

```
'dblNombre est l'argument (la valeur transmise à la fonction).
Function Multiplication(dblNombre As Double) As Double
    'Multiplie l'argument dblNombre par 2 et renvoie le résultat au nom
    'de la fonction.
    Multiplication = dblNombre * 2
End Function
```

Vous pouvez ensuite appeler cette fonction par macro.

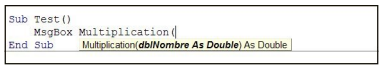
Ajoutez la procédure ci-dessous dans le module puis déclenchez l'exécution de la macro "Test".

Vba

```
Sub Test()
    'Transmet la valeur 40 à la fonction "Multiplication".
    'La fonction renvoie 80.
    MsgBox Multiplication(40)

    'Vous pouvez aussi utiliser la syntaxe suivante:
    MsgBox Multiplication(dblNombre:=40)
```

```vba
        'Vous n'avez pas besoin d'ajouter "dblNombre:=" lorsque les données sont
        'passées à la fonction dans le même ordre que la déclaration des arguments.
    End Sub
```

Lorsque vous ouvrez la parenthèse après le nom de la fonction, afin de saisir les paramètres d'appel, vous constatez que l'éditeur vous aide en affichant sous forme d'info-bulle la structure, le nombre et le type d'arguments, ainsi que le type de résultat attendu (ici, type Double).



D'ou l'importance de nommer clairement les arguments de la fonction.

Maintenant, rebasculez vers la feuille de calcul pour utiliser la même fonction dans une cellule.

Sélectionnez la cellule qui va contenir la formule.

Sélectionnez le menu Insertion/Fonction/Catégorie "Personnalisées":

Choisissez la fonction "Multiplication" dans la liste.

Indiquez une valeur pour l'argument "dblNombre".

Cliquez sur le bouton OK pour valider.

Vous pouvez aussi saisir directement la fonction dans la cellule:

=Multiplication(40)

La formule est valide si elle est placée dans le même classeur que la fonction personnelle, sinon vous devez ajouter la référence du classeur:

=NomClasseur.xls!Multiplication(40)

Une fonction est disponible uniquement lorsque le classeur qui contient la procédure est ouvert. Vous devez créer un complément pour que votre fonction personnelle soit opérationnelle dès l'ouverture de l'application. Reportez vous au chapitre **V** pour plus de détails.

Assurez-vous que type de donnée qui alimente la fonction est cohérent: Par exemple, il ne faut pas envoyer un type de donnée String vers une fonction qui attend une valeur numérique. Sinon, vous provoquez une erreur d'exécution 13 "Incompatibilité de type", ou une erreur #VALEUR! lorsque la fonction est utilisée dans une feuille de calcul.

Vérifiez aussi que le nombre d'arguments obligatoire est bien respecté. Si un élément est oublié lors de l'appel, vous provoquez une erreur de compilation "Argument non facultatif", dès le lancement de la procédure, ou une erreur #VALEUR! lorsque la fonction est utilisée dans une feuille de calcul.

Les chapitres suivants décrivent les éléments constitutifs d'une fonction.

II-B - Description de l'instruction Function

La description ci-dessous est en partie issue de l'aide Excel, agrémentée de commentaires personnels.

[Public | Private | Friend] [Static] Function name [(arglist)] [As type]

[statements]

[name = expression]

[Exit Function]

[statements]

[name = expression]

End Function

II-B-1 - Public

Public

Argument facultatif. Une fonction placée dans un module standard est de type **Public** par défaut. Il n'est donc pas nécessaire de faire précéder l'instruction **Function** par le mot clé **Public**.

II-B-2 - Private

Private

Argument facultatif. Signifie que la fonction est accessible uniquement pour les procédures VBA du module dans lequel elle est déclarée. Les fonctions déclarées **Private** (et placées dans un module standard) peuvent être utilisées dans la feuille de calcul mais n'apparaissent pas dans la boîte de dialogue "Insérer les fonctions".

II-B-3 - Friend

Friend

Argument facultatif. Est utilisé uniquement dans un module de classe. Signifie que la fonction est visible uniquement dans le projet et pas depuis des appels extérieurs.

Les classes définissent le fonctionnement des objets, sur le principe de la Programmation Orientée Objet. Un objet peut être défini par une propriété, une méthode ou un évènement.

L'emploi des classes s'adresse à des utilisateurs confirmés ayant de bonnes connaissances en programmation.

II-B-4 - Static

Static

Argument facultatif. Il indique que les variables locales de la fonction sont conservées entre les appels. L'attribut **Static** n'a pas d'effet sur les variables déclarées en dehors de la procédure Function, même si elles sont utilisées dans cette dernière. Si le mot clé Static n'est pas utilisé, la valeur des variables locales n'est pas conservée entre les appels. Attention: Les fonctions peuvent être récursives, c'est-à-dire qu'elle peuvent s'appeler elles-mêmes pour accomplir une tâche déterminée. Cette caractéristique peut entraîner un dépassement de la capacité de la pile. Le mot clé Static n'est généralement pas utilisé avec des procédures Function récursives.

II-B-5 - Name

name

Nom de la fonction. Le nom respecte les conventions standards d'affectation de noms aux variables:

* Evitez de nommer les variables en utilisant des mots clés ou instructions réservés par Excel (par exemple Val, Left...).

* Le nom des variables doit commencer par un caractère alphabétique et ne pas excéder 255 caractères. Les noms ne doivent pas contenir de caractères spéciaux. Le caractère underscore _ est accepté. Essayez de donner des noms les plus clairs possibles afin de faciliter la relecture de votre programme.

* Il est conseillé d'avoir au moins une majuscule dans la variable déclarée. Ensuite lors de la saisie de la variable en minuscule dans la macro, celle-ci reprendra automatiquement la majuscule: cette astuce permet de vérifier les fautes d'orthographe éventuelles.

* Attribuez des noms explicites pour faciliter la relecture de votre code.

Pour qu'une fonction renvoie une valeur, attribuez cette valeur au nom de la fonction (TestFonction = intCalc dans l'exemple suivant).

Vous pouvez affecter la valeur autant de fois que nécessaire et la placer n'importe où dans la procédure. Attention toutefois à ajouter des conditions de sortie si vous deviez utiliser des boucles récursives, dans le style TestFonction = TestFonction(5), sinon vous provoquerez une erreur 28 "Espace pile insuffisant".

Vba

```
Sub Essai()
    'Renvoie 2020
    MsgBox TestFonction(20)
End Sub


Function TestFonction(Arg1 As Double) As Double
    Dim intCalc As Double

    intCalc = (Arg1 * 100) + Arg1
    TestFonction = intCalc
End Function
```

Si aucune valeur n'est attribuée à l'argument **name**, la procédure renvoie une valeur par défaut.

Une fonction numérique renvoie la valeur 0.

Une fonction de type String renvoie une chaîne de longueur nulle "".

Une fonction de type Variant, la valeur Empty.

Une fonction de type Object renvoie Nothing.

Le même exemple que précédemment, renvoie 0 car aucune valeur n'est attribuée au nom "TestFonction" (mise en commentaire de la ligne TestFonction = intCalc):

Vba

```
Sub Essai()
    'Renvoie 2020
    MsgBox TestFonction(20)
End Sub

Function TestFonction(Arg1 As Integer) As Long
    Dim intCalc As Double
```

```vba
        intCalc = (Arg1 * 100) + Arg1
        'TestFonction = intCalc
    End Function
```

Si la fonction doit renvoyer une référence d'objet (par exemple Range), la donnée attribuée à l'argument **name** doit être traitée avec le mot clé **Set**:

Set DerniereCellule = .Cells(NumLigne, NumCol) dans l'exemple suivant.

```vba
    Sub Test()
        Dim Cible As Range

        Set Cible = DerniereCellule(ActiveSheet)
        MsgBox Cible.Address
    End Sub


    'Récupère la dernier cellule utilisée dans la feuille spécifiée
    '(Cet exemple ne gère pas les feuilles totalement vides)
    Function DerniereCellule(Ws As Worksheet) As Range
        Dim NumLigne As Double, NumCol As Integer

        With Ws
            NumLigne = .Cells.Find("*", .Range("A1"), , , xlByRows, xlPrevious).Row
            NumCol = .Cells.Find("*", .Range("A1"), , , xlByColumns, xlPrevious).Column

            Set DerniereCellule = .Cells(NumLigne, NumCol)
        End With
    End Function
```

II-B-6 - Exit Function

Exit Function

Cette instruction permet la sortie immédiate de la fonction. Le programme reprend à partir de la ligne qui suit l'instruction ayant appelé la fonction. Une fonction peut comporter plusieurs instructions **Exit Function**, qui peuvent être placées en n'importe quel point de la procédure.

II-B-7 - arglist

arglist

Argument facultatif. Liste de variables représentant les arguments qui sont passés à la fonction lorsqu'elle est appelée. Ces paramètres sont séparés par des virgules. Leurs noms respectent les conventions standards d'affectation de variables.

L'argument **arglist** est structuré de la manière suivante:

[Optional] [ByVal | ByRef] [ParamArray] varname[( )] [As type] [= defaultvalue]

II-B-7-a - Optional

Optional

Argument facultatif. Indique qu'un argument n'est pas obligatoire. Il est impérativement placé en fin de liste et les arguments suivants doivent également être déclarés à l'aide du mot clé **Optional** (consultez le chapitre II-E). Celui-ci ne peut pas cohabiter avec le mot clé **ParamArray**.

II-B-7-b - ByVal et ByRef

ByVal

Argument facultatif. Permet de passer à une procédure la valeur d'un argument plutôt que son adresse. La procédure peut de ce fait accéder à une copie de la variable. La valeur réelle de cette dernière n'est donc pas modifiée par la procédure à laquelle elle est passée. Si la procédure appelée change la valeur des variables, elles ne changeront pas dans la procédure appelante. L'utilisation de **ByVal** implique un temps de calcul plus long et nécessite un espace mémoire plus important.

ByRef

Argument facultatif. Permet de passer à une procédure l'adresse d'un argument plutôt que sa valeur. La procédure peut ainsi accéder à la variable proprement dite. La valeur réelle de cette dernière peut, de ce fait, être modifiée par la procédure à laquelle elle a été passée. Par défaut, les arguments sont passés par référence. Si la procédure appelée change la valeur de ces variables, elles changeront au retour dans la procédure appelante.

Nota:

Vous n'avez pas besoin de spécifier **ByVal** ou **ByRef** si vos fonctions servent exclusivement dans la feuille de calcul car une formule ne peut pas agir sur les données appelantes.

Un exemple de mise en application pour montrer les valeurs prises successivement par la variable 'Donnee', en fonction du passage par des sous procédures ByRef, ByVal, puis non spécifiée.

Vba

```
Option Explicit

Sub Test()
```

Vba
```
        Dim Donnee As Integer

        Donnee = 50

        MaProcedure_1 Donnee
        MsgBox Donnee

        MaProcedure_2 Donnee
        MsgBox Donnee

        MaProcedure_3 Donnee
        MsgBox Donnee
    End Sub


    'Passe la référence en argument.
    Function MaProcedure_1(ByRef x As Integer) As Integer
        x = x * 2
    End Function


    'Passe la valeur en argument.
    Function MaProcedure_2(ByVal y As Integer) As Integer
        y = y * 2
    End Function


    'ByRef est la valeur par défaut si non spécifiée.
    Function MaProcedure_3(z As Integer) As Integer
        z = z * 2
    End Function
```

II-B-7-c - ParamArray

ParamArray

Argument facultatif. Est utilisé uniquement comme dernier argument de **arglist** pour indiquer que le dernier argument est un tableau **Optional** d'éléments de type Variant. Le mot clé **ParamArray**, qui permet d'indiquer un nombre quelconque d'arguments, ne peut être utilisé avec les mots clés ByVal, ByRef ou Optional. Consultez le chapitre II-D.

II-B-7-d - varname

varname

Indique le nom de la variable représentant l'argument. Il respecte la convention standard d'affectation de noms aux variables.

II-B-7-e - Type

type

Argument facultatif. Type de données de l'argument passé à la procédure : Byte, Boolean, Integer, Long, Currency, Single, Double, Date, String (de longueur variable uniquement), Object, Variant. Si le paramètre n'est pas **Optional**, un type défini par l'utilisateur peut également être indiqué.

II-B-7-f - defaultvalue

defaultvalue

Argument facultatif. Représente une constante ou expression constante. Cet argument est valide uniquement pour les paramètres **Optional**. S'il s'agit d'un type Object, seul Nothing peut être explicitement indiqué en valeur par défaut. Cet argument permet de définir une valeur par défaut pour un paramètre optionnel, dans le cas ou l'utilisateur n'aurait rien indiqué lors de l'appel de procédure (consultez le chapitres II-E).

La syntaxe est:

Optional [NomVariable] As [TypeVariable] = [Donnée par défaut]

Un exemple:

Vba

```
Sub Test()
    'Renvoie tous les arguments spécifiés
    MsgBox MaProcedure(5, "fifi")
    'Renvoie 5 et "mimi" car l'argument Optional (strPrenom) n'est pas précisé
    MsgBox MaProcedure(5)
End Sub


Function MaProcedure(lngValeur As Long, Optional strPrenom As String = "mimi")
    MaProcedure = lngValeur & vbCrLf & strPrenom
End Function
```

II-B-8 - Type

type

Argument facultatif. Type de donnée renvoyé par la fonction. La fonction peut être de type Byte, Boolean, Integer, Long, Currency, Single, Double, Date, String, Object, Variant, ou tout type défini par l'utilisateur.

Le typage respecte les conventions standards d'affectation de **variables**.

II-B-9 - statements

statements

Argument facultatif. Représente le groupe d'instructions à exécuter dans la fonction.

II-B-10 - expression

expression

Argument facultatif. Représente la valeur renvoyée par la fonction.

II-B-11 - Le nom des variables

Les variables explicitement déclarées dans une fonction sont toujours locales au niveau de la procédure.

Une fonction utilisant une variable qui n'est pas explicitement déclarée peut provoquer un conflit si une variable définie dans une autre procédure possède le même nom.

Pour éviter ce genre de conflits, déclarez explicitement les variables en utilisant l'instruction **Option Explicit**.

Cette instruction doit apparaître tout en haut dans le module, avant toute procédure. Lorsque l'instruction est utilisée, un message d'erreur identifie toute variable non définie ou mal orthographiée.

Pour qu'Option Explicit s'insère automatiquement dans chaque nouveau classeur:

Allez dans l'éditeur de macros.

Menu Outils

Options

Dans l'onglet "Editeur", cochez l'option "Déclaration Explicite des variables".

II-C - La méthode Application.Volatile

Ajoutez la ligne d'instruction **Application.volatile** en début de fonction lorsque celle-ci est utilisée dans la feuille et doit être recalculée automatiquement à chaque fois qu'une modification ou un calcul est effectué dans une cellule quelconque du classeur.

II-D - L'argument ParamArray

Les tableaux de paramètres (ParamArray) permettent de passer un tableau d'arguments à une procédure. Le nombre d'éléments indiqué peut être variable. Vous pouvez de cette manière passer plusieurs informations (et dont le nombre peut varier d'un appel à l'autre) à une fonction.

L'argument tableau ParamArray:

* Doit obligatoirement être déclaré en type Variant.

* Ne peut pas coexister avec des arguments Optional.

* Doit être placé en dernière position dans la liste des arguments.

* Les arguments ne sont pas facultatifs.

* Chaque argument peut être d'un type de donnée différent.

* Le nombre d'arguments n'est pas limité.

* L'indice inférieur du tableau est toujours 0.

Pour détecter un argument **ParamArray** vide, effectuez un test qui déterminera si la limite maximale **du tableau** est inférieure à sa limite minimale.

Lorsque l'appel de procédure est effectué, chaque argument fourni dans l'appel devient un élément indexé du tableau.

Un exemple qui supprime une série de caractères dans une chaîne:

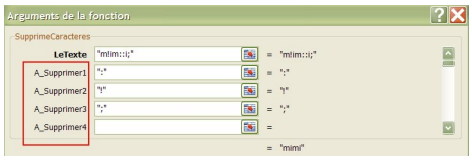
Les caractères ";", ":", "!" sont passés au tableau ParamArray "A_Supprimer1". La procédure boucle ensuite sur les éléments du tableau et supprime chaque caractère trouvé dans la chaîne "LeTexte".

Vba

```
Option Explicit

Sub Test()
    Dim strChaine As String

    strChaine = "m!im::i;"
    'La procédure va supprimer les caractères ; : !
    'dans la variable "Chaine"
    strChaine = SupprimeCaracteres(strChaine, ";", ":", "!")
    MsgBox strChaine
End Sub

Function SupprimeCaracteres(LeTexte As String, ParamArray A_Supprimer1())
    Dim i As Integer
    'Boucle sur les éléments du tableau
    For i = 0 To UBound(A_Supprimer1())
```

Vba
```
            'Supprime les caractères spécifiés
            LeTexte = Replace(LeTexte, A_Supprimer1(i), "")
        Next i
        SupprimeCaracteres = LeTexte
    End Function
```

Si vous utilisez **ParamArray** dans une fonction, ajoutez le chiffre 1 en fin de nom du paramètre. Ainsi, l'assistant de fonctions Excel incrémentera automatiquement chaque nouvel élément indiqué par l'utilisateur.



Ce deuxième exemple crée un tableau d'une dimension, à partir des données passées dans la fonction.

Une fonction qui retourne un tableau doit impérativement être déclarée en type Variant.

Vba
```
    Function CreationTableau(ParamArray Cellules1()) As Variant
        'Adapté de:
        'http://support.microsoft.com/?kbid=213403
        '
        Dim VarTab() As Variant
        Dim Temp As Variant
        Dim i As Integer
        Dim w As Integer, X As Integer, y As Integer, z As Integer

        i = 1

        'Boucle sur les éléments du tableau de paramètres.
        For X = 0 To UBound(Cellules1)
            If TypeName(Cellules1(X)) = "Range" Then
                Set Temp = Cellules1(X)
                'Vérifie si le paramètre passé à la fonction est une cellule simple
                'ou une plage.
                If IsArray(Temp) Then
                    'Intègre chaque cellule de la plage dansle tableau.
                    For y = 1 To UBound(Temp.Value)
                        For z = 1 To UBound(Temp.Value, 2)
                            'Permet de filtrer les cellules vides.
                            'If Not IsEmpty(Temp(y, z).Value) Then
                                ReDim Preserve VarTab(1 To i)
                                VarTab(i) = Temp(y, z).Value
                                i = i + 1
                            'End If
                        Next z
                    Next y
                    Else
```

```vba
                    'Permet de filtrer les cellules vides.
                    'If Not IsEmpty(Temp) Then
                        'Intègre la cellule dans le tableau.
                        ReDim Preserve VarTab(1 To i)
                        VarTab(i) = Temp
                        i = i + 1
                    'End If
                End If
            Else
                ReDim Preserve VarTab(1 To i)
                VarTab(i) = Cellules1(X)
                i = i + 1
            End If
        Next X

        CreationTableau = VarTab
    End Function
```

Vous pouvez indiquer des cellules uniques (E1), des plages (A1:A10, C1:C10) ou des données (80) lorsque vous appelez la fonction:

```vba
    Sub Test()
        Dim Tb As Variant, xTab As Variant

        Tb = CreationTableau(Range("A1:A10"), Range("C1:C10"), Range("E1"), 80)

        '--- Vérifie si le tableau est vide ---
        On Error Resume Next
        'xTab va prendre la valeur Empty si le tableau est vide.
        xTab = UBound(Tb)
        On Error GoTo 0

        'Renvoie le nombre d'éléments du tableau
        If Not IsEmpty(xTab) Then MsgBox UBound(Tb)
    End Sub
```

II-E - Les arguments optionnels

Les fonctions sont constituées d'arguments obligatoires et optionnels. Comme leur nom l'indique, les arguments obligatoires sont nécessaires au bon fonctionnement de la procédure. Quant aux arguments optionnels, lorsqu'ils sont omis, cela n'empêche pas l'exécution de la macro. Les arguments optionnels sont obligatoirement placés en fin de la déclaration.

Voici un exemple de fonction à placer dans un module standard. L'argument **Coeff** est déclaré **Optional**:

La fonction **MaFonctionPerso** renvoie une donnée de type Double.

Vba

```
Function MaFonctionPerso(NbValeurs As Long, Optional Coeff As Variant) As Double

    'Spécifie que la fonction doit être recalculée à chaque fois qu'un calcul
    'est effectué dans une cellule quelconque de la feuille.
    Application.Volatile

    'Vérifie si l'argument optionnel à été indiqué
        'Nota: Coeff doit être impérativement de type Variant.
    If IsMissing(Coeff) Then
        'S'il n'y a pas d'argument optionnel:
        MaFonctionPerso = (NbValeurs + 3)
    Else
        'S'il y a un argument optionnel:
        MaFonctionPerso = (NbValeurs + 3) * Coeff
    End If

End Function
```

Vous pouvez ensuite appeler cette fonction depuis une procédure Sub.

Vba

```
Sub AppelFonction()
    MsgBox MaFonctionPerso(10)
    MsgBox MaFonctionPerso(10, 5)
End Sub
```

Mais aussi l'utiliser depuis la feuille de calcul:

Formule

```
=MaFonctionPerso(10)
=MaFonctionPerso(10;5)
```

Formule

```
'Si le premier argument fait référence à la cellule A1:
=MaFonctionPerso(A1;5)
```

Rappel:

Il est possible de définir une donnée par défaut pour un argument de type **Optional**. Il s'agit de la valeur qui sera prise en compte si l'utilisateur n'a pas spécifié l'argument optionnel.

La valeur par défaut est définie de la manière suivante:

Optional [NomVariable] As [TypeVariable] = [Donnée par défaut]

Vba

```
Sub Test()
    'Renvoie tous les arguments spécifiés
    MsgBox MaProcedure(5, "fifi")
    'Renvoie 5 et "mimi" car l'argument Optional (strPrenom) n'est pas précisé
    MsgBox MaProcedure(5)
End Sub

Function MaProcedure(lngValeur As Long, Optional strPrenom As String = "mimi") As Variant
    MaProcedure = lngValeur & vbCrLf & strPrenom
End Function
```

Il est conseillé d'utiliser un type Variant pour les arguments Optionnels. En effet, vous pourrez ensuite intégrer la fonction **IsMissing** dans votre procédure afin de vérifier si l'argument a été spécifié ou non.

La fonction **IsMissing** permet de vérifier si des arguments optionnels sont indiqués lors d'un appel. **IsMissing** renvoie la valeur True si l'argument n'a pas été spécifié, sinon elle renvoie la valeur False.

Si vous devez utiliser cette fonction, les arguments optionnels doivent impérativement être de type Variant. En effet, une variable Variant prend valeur "Empty" par défaut (vide) et peut donc être identifiée par la fonction **IsMissing**.

Vba

```
Sub test()
    NomFonction Arg1:=0, Arg2:=56765765
    NomFonction Arg1:=453
    NomFonction Arg2:=56765765
    NomFonction
End Sub


Function NomFonction(Optional Arg1 As Variant, Optional Arg2 As Variant) As Variant
    MsgBox "Arguments manquants:" & vbCrLf & _
        IsMissing(Arg1) & " / " & IsMissing(Arg2)
End Function
```

II-F - La gestion des erreurs

Nous avons vu dans les chapitres précédents qu'un type de donnée incohérent entre les paramètres d'appel, les arguments et les variables de la fonction, provoque une erreur d'exécution.

Une solution palliative peut consister à créer tous les arguments en type Variant et intégrer une vérification à l'intérieur de la fonction. Il existe des fonctions prévues à cet effet:

* TypeName

* VarType

* IsDate

* IsNumeric

* IsEmpty

* IsMissing

* IsArray

* IsObject

* IsNull

La méthode **Raise**, utilisée dans l'exemple suivant, gère une erreur d'exécution dans la macro, si le type de donnée ne correspond pas ce qui est attendu.

La procédure affiche un MsgBox qui reprend des informations d'aide personnalisées que vous aurez préalablement défini. La méthode **Raise** donne aussi accès à un fichier d'aide .chm (bouton Aide dans le MsgBox). Placez la procédure "PerimetreCercle" dans un module standard, puis la formule suivante dans une cellule de la feuille:

=PerimetreCercle("mimi")

Une erreur va survenir car un argument de type texte (mimi) est indiqué à la place d'une donnée numérique. Un message d'alerte va indiquer que le type de donnée ne convient pas à procédure. La fonction **IsNumeric** permet d'effectuer cette vérification.

Vba

```
Function PerimetreCercle(Diametre As Variant) As Variant

    'Spécifie que la fonction doit être recalculée à chaque fois qu'une modification
    'est effectuée dans une cellule quelconque de la feuille.
    Application.Volatile


    On Error GoTo Fin
    'Affiche un message d'alerte si l'argument (Diametre) n'est
    'pas correctement saisi. (Si par exemple vous saisissez un texte à la place
    'd'un nombre).
    If Not IsNumeric(Diametre) Then _
        Err.Raise 5010, , vbCrLf & "Attention!" & vbCrLf & _
            "Vous devez saisir une valeur numérique." & vbCrLf & _
            "La donnée '" & Diametre & "' n'est pas valide.", _
            "C:\dossier\fichier d'aide.chm", 9900

    PerimetreCercle = Diametre * 3.14
Exit Function
```

Vba

```
    Fin:

    'Affiche un message correspondant aux paramètres indiqués dans la
    'méthode Raise.
    MsgBox "Erreur: " & Err.Number & vbCrLf & _
        Err.Description, vbExclamation + vbMsgBoxHelpButton, , Err.HelpFile, Err.HelpContext

    'Force la valeur d'erreur #N/A dans la cellule, si le type de données est erroné.
    PerimetreCercle = CVErr(xlErrNA)

    End Function
```

Remarque:

Si le contenu de la formule n'est pas ensuite corrigé, le message d'erreur continuera de s'afficher systématiquement à chaque modification ou re-calcul dans la feuille.

III - Description personnalisée et catégorie particulière

Lorsque vous utilisez vos fonctions dans une cellule, celles ci sont disponibles depuis la catégorie "Personnalisées", par défaut. Aucune description n'apparait dans la fenêtre "Arguments de la fonction" contrairement aux fonctions natives de l'application.

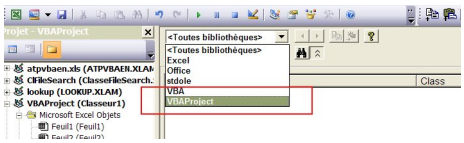
Le détail de chaque variable ne peut pas être commenté dans la fenêtre "Arguments de la fonction". Il vous appartient donc de nommer les paramètres de manière le plus explicite possible.

Par contre vous pouvez ajouter une description générale:

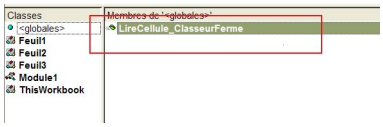
Dans l'éditeur de macros:

Utilisez le raccourci clavier **F2** pour afficher l'explorateur d'objets.

Sélectionnez "VBAproject" dans le menu déroulant.



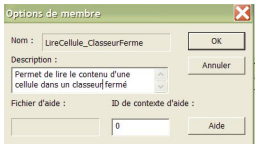
Votre fonction apparait dans la fenêtre de droite.



Faites un clic droit.

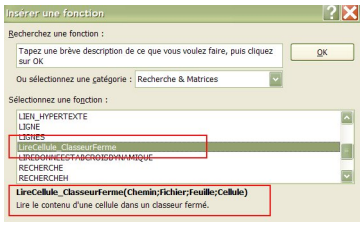
Sélectionnez l'option "Propriétés".

Saisissez la description.



Cliquez sur le bouton OK pour valider.

Vous pouvez visualiser le résultat en affichant la boîte de dialogue "Insérer une fonction":



Une autre solution, en passant pas la boîte de dialogue des macros:

Menu Outils

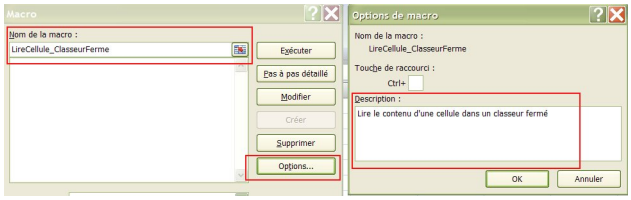
Macro

Macros

Dans le champ "Nom de la macro", saisissez le nom complet de la fonction.

Le bouton " Options" est alors accessible pour ajouter la description.

Cliquez sur le bouton OK pour valider.



Cette action peut aussi être réalisée par code VBA, ainsi que le placement de la fonction dans une catégorie spécifique:

Cet exemple ajoute une description et place la fonction dans la catégorie 5 (Recherche et matrices).

La macro n'a besoin d'être lancée qu'une seule fois. Les paramètres seront ensuite définitivement enregistrés dans le classeur.

Vba

```
Application.MacroOptions Macro:="LireCellule_ClasseurFerme", _
    Description:="Lire le contenu d'une cellule dans un classeur fermé", Category:=5
```

La liste des catégories (Excel2007):

1 Finances

2 Date et Heure

3 Math et Trigo

4 Statistiques

5 Recherche Matrices

6 Base de données

7 Texte

8 Logique

9 Informations

10 Commandes

11 Personnalisation

12 Contrôle de macros

13 DDE/Externe

14 Personnalisées

15 Ingénierie

16 Cube

Les catégories 10, 12 et 13 sont visibles uniquement si elles contiennent une fonction.

IV - Un fichier d'aide pour vos fonctions personnelles

La méthode **MacroOptions** possède une option pour attacher un fichier d'aide à une fonction personnelle.

Placez la fonction complémentaire dans un module standard:

Vba

```
Function NomFonction(Arg1 As String) As String
'...
End Function
```

Vous allez maintenant utiliser la macro "Test" pour attacher un fichier d'aide et son identificateur de contexte.

Vba

```
Sub Test()
'7 correspond à la catégorie "Texte"
'"C:\mon fichier d'aide.CHM" est un fichier d'aide stocké sur votre disque dur.
'4050 est l'identificateur de contexte permettant d'accéder directement à la page d'aide
    'de votre fonction.
Application.MacroOptions Macro:="NomFonction", Category:=7, _
    Description:="La description de ma fonction complémentaire.", _
    HelpFile:="C:\mon fichier d'aide.CHM", _
    HelpContextID:=4050
End Sub
```

Ensuite, appelez la fonction dans une cellule:

Sélectionnez la catégorie "Texte" et recherchez votre fonction complémentaire dans la liste.

Cliquez sur le lien "Aide sur cette fonction" pour afficher le fichier d'aide que vous avez précédemment attaché.

Nota:

A ce jour, la fonctionnalité semble ne plus être opérationnelle sous excel2007.

V - Rendre une fonction opérationnelle dès l'ouverture d'Excel

Une fonction est disponible uniquement lorsque le classeur qui contient la procédure est ouvert. Vous devez créer un complément pour que votre fonction personnelle soit disponible dès l'ouverture de l'application.

Créez un nouveau classeur vierge.

Insérez un module standard depuis l'éditeur de macros.

Copiez-y les fonctions de votre choix.

Sauvegardez le fichier au format .xlam pour Excel2007 et .xla pour les versions antérieures d'Excel.

Vous remarquerez que la boîte de dialogue s'ouvre automatiquement sur le répertoire spécifique des macros complémentaires, lorsque vous choisissez ces types d'extension.

Ce chemin est généralement:

C:\Documents and Settings\nom_utilisateur\Application Data\Microsoft\Macros complémentaires (ou AddIns)

Nommez votre complément.

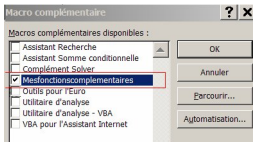
Cliquez sur le bouton OK pour valider.

Pour activer la macro complémentaire (avant Excel2007):

Menu Outils

Macros complémentaires

Si vous avez bien enregistré le classeur dans le répertoire spécifique des macros complémentaires, votre fichier doit apparaitre dans la boîte de dialogue qui s'affiche à l'écran (Cliquez sur le bouton "Parcourir" si votre complément n'apparait pas dans la liste).



Cochez la ligne correspondante.

Cliquez sur le bouton OK pour valider.

Refermez puis ré-ouvrez Excel.

Faites des essais d'utilisation. Vous constatez que vos fonctions sont désormais automatiquement disponibles.

Sous Excel2007:

Cliquez sur le bouton "Office".

Cliquez sur le bouton "Options Excel".

Sélectionnez le menu Compléments.

Choisissez "Compléments Excel" dans le menu déroulant "Gérer" (en bas de la fenêtre).

Cliquez sur le bouton "Atteindre".

Remarque :

Lorsque vos classeurs utilisent des fonctions issues d'un complément, et doivent être transmis à d'autres utilisateurs, assurez vous que ceux-ci disposent aussi du complément. Sinon, les formules placées dans la feuille de calcul ne seront plus opérationnelles et renverront une valeur d'erreur #NOM?.

VI - Liens

Présentation de l'éditeur de macros.

Les variables.

Les variables tableaux.

La gestion des erreurs dans Excel.

Créer un fichier d'aide de type .chm

VII - Remerciements

Je remercie toute l'équipe Office, et particulièrement **Maxence Hubiche**, **Ouskel'n'or** pour la relecture et la correction du tutoriel.

VIII - Téléchargement