VBA pour Excel

L'essentiel ... pour démarrer !

1 Généralités

Visual Basic pour Applications (VBA) est un environnement de développement calqué sur Visual Basic, un outil de développement d'applications Windows. Tandis que **les programmes Visual Basic (VB) sont autonomes, les programmes VBA ne peuvent être exécutés qu'à partir d'une application** intégrant cet environnement de développement (Excel ou une autre application). Les programmes VBA sont donc attachés à un document Word, une feuille de calcul Excel et constituent un projet.

L'enregistrement de macros constitue une bonne initiation à VB. **L'enregistreur de macros mémorise chacune des actions réalisées par l'utilisateur puis les traduit en instructions VB**. Il suffit d'exécuter ensuite la macro pour répéter l'ensemble des actions ainsi enregistrées.

Si **certaines instructions sont spécifiques à l'application** (pour Excel par exemple, les instructions permettant d'affecter une formule à une cellule), **d'autres sont communes à l'ensemble des applications Office** (affichage des boîtes de dialogue pour permettre une interaction de l'utilisateur influant sur le déroulement de la macro, structures de contrôle permettant de réaliser des boucles...) et permettent de créer des macros évoluées qui, attachées aux documents manipulés, constituent de véritables applications répondant à des besoins spécifiques

VB Editor est l'environnement de développement intégré des applications Office. Il permet de visualiser, de gérer les projets VBA, d'écrire, de modifier et de déboguer les macros existantes.

1.1 Les Objets de VBA

VBA est un langage de programmation orienté objet, c'est à dire qu'il manipule des objets de l'application en cours. Un **objet** est caractérisé par un **nom** et possède des **propriétés**, on peut lui appliquer des **méthodes** pour modifier son comportement. Ainsi toute feuille de calcul renvoie à **la classe** sheets (cette classe définie les propriétés associées à toute feuille de calcul et les méthodes qui y sont applicables).Une collection désigne l'ensemble des **occurrences** (ou **instances**) d'un objet : la collection Workbooks regroupe l'ensemble classeurs ouverts, la collection sheets toutes les feuilles d'un classeur.

1.2 Les objets sont hiérarchisés

Par exemple classeur ➔ feuille ➔ cellule

1.2.1 Accéder aux objets

Pour accéder aux objets, il est nécessaire de spécifier le chemin à emprunter (le point est utilisé comme séparateur des différents objets composant le chemin. Ainsi pourra-t-on distinguer la feuille2 du classeur 1 de la feuille2 du classeur2 lorsque l'on souhaitera les rendre actives :

workbooks("classeur1").sheets("feuil2").activate (la méthode "activate" est appliquée à la feuille2 du classeur1).

workbooks("classeur2").sheets("feuil2").activate (la méthode "activate" est appliquée à la feuille2 du classeur2).

Bien entendu, si l'on travaille dans un seul classeur, il sera inutile de préciser l'identité du classeur contenant la feuille et dans ce cas *sheets("feuil2").activate* suffira.

1.2.2 Modifier les objets

Comme nous l'avons évoqué supra, les objets ont des propriétés. Les occurrences de ces objets se distinguent entre elles par les valeurs associées à ces propriétés. Ces valeurs peuvent renvoyer à différents types :

- chaîne de caractères,
- valeur numérique
- valeur booléenne (true ou false)
- constante(se présentant sous forme de chaîne de caractères, mais correspondant à des valeurs numériques)

Dans le code Visual Basic, on doit identifier un objet avant de pouvoir changer la valeur de l'une de ses propriétés ou lui appliquer une de ses méthodes (voir infra).

Pour modifier une valeur d'une propriété on procède par affectation (objet.propriété=valeur)

Sheets("feuille1").name="Résultats" (renomme la feuille1 en Résultats)

ActiveCell.Value=5 (la propriété "value" de la cellule active reçoit la valeur 5)

1.2.3 Les méthodes

Une méthode permet de modifier une propriété d'une occurrence d'objet. Ainsi par exemple, la méthode Select rend la feuille1 active : Sheets.('feuille1").Select

1.2.4 Les événements

Un événement est une action reconnue par un objet. La programmation événementielle est une technique qui consiste à réaliser un ensemble de procédures qui seront déclenchées par des objets appelés à juste titre **déclencheur**. Un bouton est un exemple type de déclencheur.

Par exemple un clic sur le bouton 

effacera le contenu de cellules.

1.2.5 Les fonctions

Les fonctions servent à renvoyer une information, selon les éléments qui leurs sont fournis. VBA fournit des fonctions en standard mais il est possible de construire ses propres fonctions(voir 3.2.2).

2 Les premières macros

Comme nous l'avons dit précédemment, l'enregistrement de macros constitue une bonne approche pour apprendre VBA car elles génèrent du code, c'est à dire le texte qu'il aurait fallu saisir en VBA.

2.1 Création d'une macro

1. Ouvrir le classeur "Sofres99.xls"
   Soit à mettre en gras les intitulés de colonnes sur la plage B5:E5
2. Activer la commande Outils/macros/Nouvelle Macro et renseigner le nom de la macro dans la boîte de dialogue :

Remarque : il est possible de saisir une description de la macro et de lui attribuer un raccourci-clavier pour l'exécuter plus rapidement.

3. Procéder à la mise en gras manuellement et terminer en cliquant sur le bouton d'arrêt d'enregistrement :

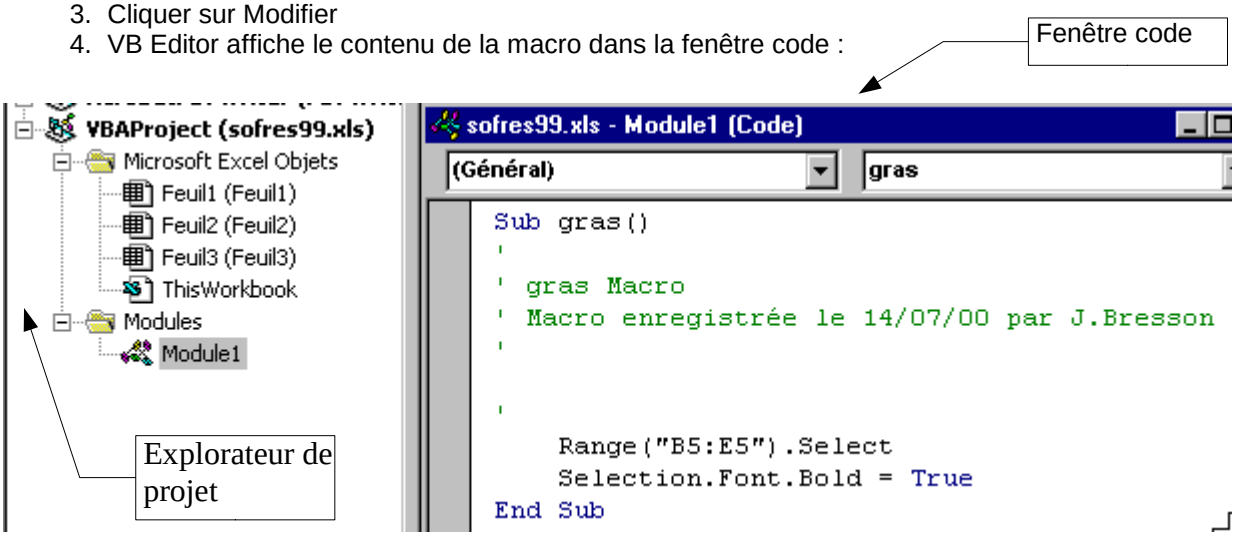
4. Remettre en non gras les cellules et tester la macro
   Activer pour cela la macro par la commande : outils/macros/macros et exécuter le macro "gras"

5. Constater l'effet de l'action de la macro.

2.2 Structure de la macro

Lors de l'enregistrement de la macro, les actions effectuées ont été codée en VB. On peut visualiser le code généré.

1. Choisir Outils/macro/macros
   (ou directement Visual Basic Editor mais il faudra identifier la bonne macro s'il y en a plusieurs)
2. Sélectionner la macro "gras"
3. Cliquer sur Modifier
4. VB Editor affiche le contenu de la macro dans la fenêtre code :



La structure est la suivante :

Sub nom_du_sous_programme()

' les commentaires sont précédés d'apostrophes

' Couleur verte pour les commentaires et couleur bleue pour les mots clés du langage

Suivent des instructions

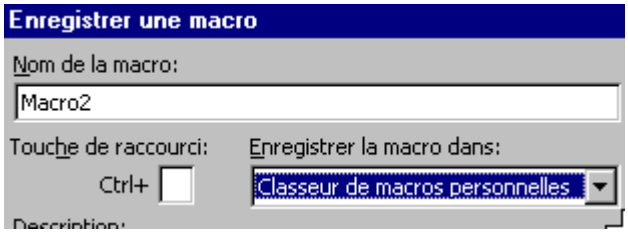
Suivent des instructions

End sub

Exercice :
Dans le classeur Sofres99.xls, enregistrer une macro sélectionne la plage A5:E13 et qui génère un graphique (barres horizontales) dans une nouvelle feuille graphique nommée Missions.

2.3 Lieux de stockage des macros

Dans l'exemple précédent la macro a été enregistrée dans le classeur courant, mais il est possible de la rendre utilisable depuis tout classeur, en demandant son enregistrement dans le classeur "perso.xls" lui même stocké dans le dossier office/xlstart (ou xlouvrir) de la machine de l'utilisateur.

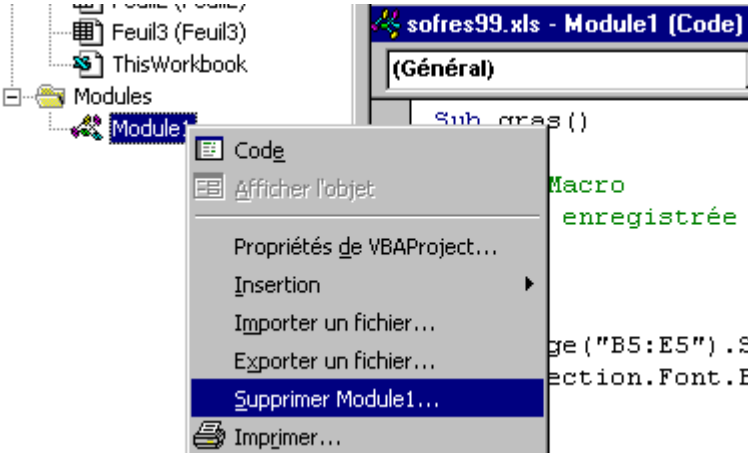


Il se peut également que l'on souhaite pouvoir diffuser ses applications avec des macros stockées dans un classeur séparé. On choisira alors l'option d'enregistrement "macro complémentaire". Celles-ci seront alors stockées dans un classeur .XLA (voir l'aide en ligne de VBA sur ce thème). On crée ses macros dans un nouveau classeur qui est ensuite enregistré au format XLA. Ensuite, pour tout classeur faisant appel à ces macros, on active la commande : outils/macro complémentaire / parcourir, on sélectionne la feuille xla, toutes ses macros sont alors actives dans le classeur courant.

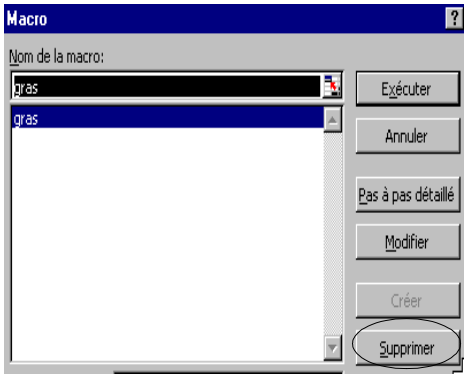
Attention, pour pouvoir consulter l'aide en ligne VBA pour Excel il faut avoir coché l'option correspondante au moment de l'installation d'Office, il faudra relancer l'installation et cocher l'option pour pouvoir en bénéficier, seule l'aide VBA est alors rajoutée, Office n'est en rien réinstallé intégralement... heureusement !).

2.4 Suppression d'une macro, d'un module

- Depuis VB Editor en activant le menu contextuel dès lors que le module 1 est pointé dans l'explorateur de projet



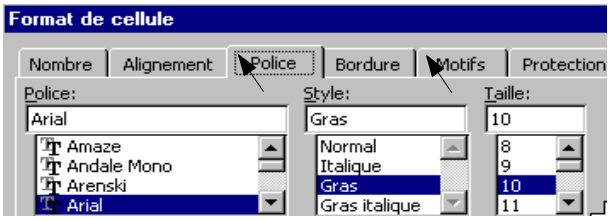
- Ou depuis la feuille de calcul Excel
  Outils/macro/macros et cliquer sur le bouton "supprimer"



2.5 Variante 1 : Mettre en gras les en-tête de colonnes et appliquer une trame grise

Provoquer l'enregistrement de la macro comme précédemment...

On utilisera la commande Format/cellules pour choisir successivement l'alignement, la police en gras et la couleur de fond.



Terminer l'enregistrement de la macro et visualiser le code du sous programme *Gras_trame_grise()* généré sous VB Editor

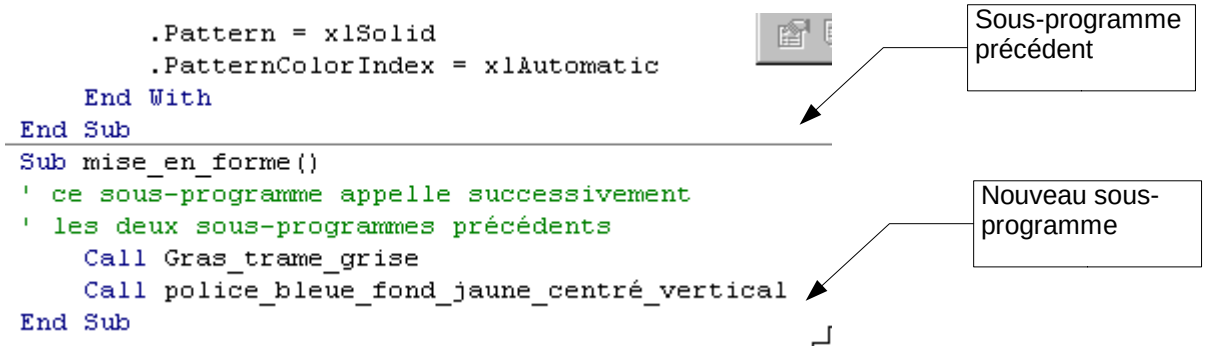
Les instructions With et End With encadrent l'ensemble des propriétés des objets Font puis Interior.

```
Sub Gras_trame_grise()
' Gras_trame_grise Macro
' Macro enregistrée le 14/07/00 par J.Bresson
    Range("B5:E5").Select
    With Selection.Font
        .Name = "Arial"
        .FontStyle = "Gras"
        .Size = 10
        .Strikethrough = False
        .Superscript = False
        .Subscript = False
        .OutlineFont = False
        .Shadow = False
        .Underline = xlUnderlineStyleNone
        .ColorIndex = xlAutomatic
    End With
    With Selection.Interior
        .ColorIndex = 48
        .Pattern = xlSolid
        .PatternColorIndex = xlAutomatic
    End With
End Sub
```

Exercice : Générer une nouvelle macro permettant de d'afficher les libellés en bleu sur fond jaune
Dans VB Editor on remarquera que le module 1 s'est enrichi d'un nouveau sous-programme :
police_bleue_fond_jaune_centré_vertical()

2.6 Variante 2 : Associer deux sous-programmes

1. Dans VB Editor, il suffit de déposer le point d'insertion à la fin du dernier sous-programme du module 1 et de générer un nouveau sous-programme en saisissant Sub nom du sous-programme suivi de ().



2. Une remarque précisant le rôle de ce sous-programme est ensuite ajoutée derrière l'apostrophe qui précise qu'il s'agit d'une remarque.
3. Suivent les deux instructions qui appellent successivement les deux sous-programmes précédents (Call appelle le sous-programme, peut être omis mais facilite la lecture du code, attention ne pas mettre de parenthèses)
4. Enfin End Sub clôt le sous-programme.

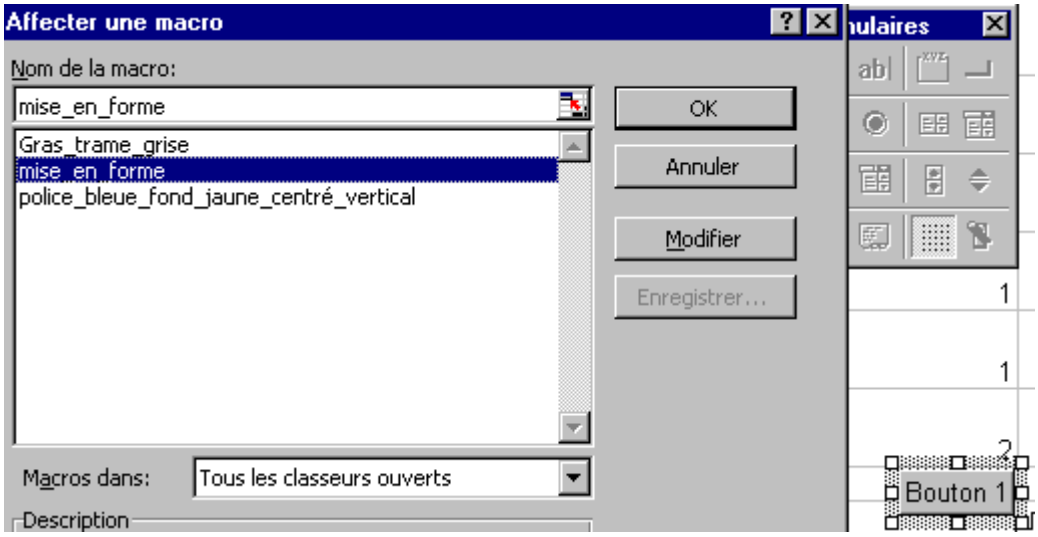
En revenant sur la feuille de calcul, la commande Outils/macro/macros permet de constater qu'une nouvelle macro a été réalisée.

2.7 Association d'une macro à un événement

1. Depuis la feuille Excel, activer la commande Affichage/barre d'outils/formulaires
   On dispose alors d'un ensemble d'outils permettant de créer des contrôles dans une feuille de calcul

2. Activer l'outil "bouton" et cliquer dans la feuille de calcul sous le tableau de chiffres

3. La boîte de dialogue "affecter une macro" s'ouvre et propose d'associer une macro à ce bouton (par défaut un clic sur le bouton provoquera l'exécution de la macro choisie (ici "mise_en_forme").



4. Il reste à modifier le texte du bouton : pour cela on sélectionne le bouton en gardant la touche <Ctrl> activée (afin d'éviter l'exécution de la macro) puis on modifie le texte du bouton en déposant le point d'insertion dans le texte du bouton.

Remarque : Pour faire apparaître la commande d'exécution de macro sous forme d'un bouton dans la barre d'outils : **Affichage /barre d'outils / personnaliser / commande / formulaire** : faire alors glisser le bouton dans la barre de menu. Ensuite il est nécessaire de faire un clic-droit sur le bouton généré et de choisir la commande **Affecter une macro** (éventuellement, on peut modifier l'image attachée à la macro en choisissant « modifier l'image du bouton » dans le même menu contextuel).

2.7.1 L'intérêt du mode enregistrement

Dans cette première approche, nous avons vu comment enregistrer une macro, accéder à son code dans VB Editor, créer soi-même un petit sous-programme et affecter une macro à un bouton.

Ces enregistrements sont utiles car ils génèrent un code "chargé de syntaxe", qu'il n'y a donc pas à retenir (mise en couleur des cellules d'une plage par exemple). Ils permettent par ailleurs de se familiariser progressivement avec le code Visual Basic généré.

Ces pratiques sont suffisantes pour toutes les tâches répétitives dont les paramètres sont connus à l'avance (nombre de cellules concernées par exemple). Ainsi, dans un classeur contenant de multiples feuilles de calcul, il sera judicieux de prévoir une feuille ne contenant que des boutons et donnant accès à toutes les autres tout en explicitant leurs contenus. On pourra créer facilement des petites interfaces qui rendent plus conviviale l'utilisation des feuilles d'un classeur.

2.7.2 Les limites des macros enregistrées

Mais on touche rapidement la limite des macros en mode enregistrement dès lors que l'on souhaite balayer un tableau pour faire des recherches ou offrir la possibilité à l'utilisateur d'intervenir pendant l'exécution de la macro, de saisir des informations et d'en modifier le cours.

Ces deux nouvelles approches des macros seront envisagées dans les chapitres suivants à partir d'exemples simples.

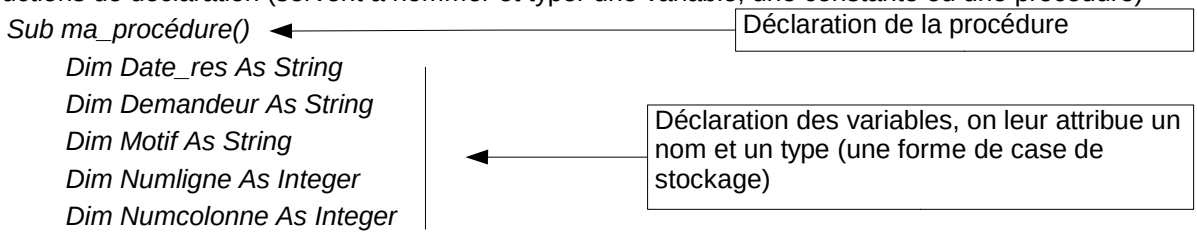
3 La structure d'un programme en Visual Basic

Dans l'exemple précédent on a pu remarquer qu'un module pouvait contenir plusieurs sous-programmes (suites d'instructions comprises entre Sub et End Sub, appelées procédures), certains pouvant en appeler d'autres.

Le fait de "découper" un programme complexe en procédures permet de mobiliser telle ou telle procédure dans différentes parties du programme. Le programme est plus lisible et le code associé à une procédure n'est ainsi jamais saisit deux fois.

3.1 Les instructions composant une procédure relèvent de trois types

- Instructions de déclaration (servent à nommer et typer une variable, une constante ou une procédure)



- *Instructions d'affectation (affectation d'une valeur à une variable, à une constante, à une propriété)*

  Pour modifier une valeur d'une propriété on procède par affectation (**objet.propriété=valeur**)

  Sheets("feuille1").name="Résultats" (renomme la feuille1 en Résultats)

  ActiveCell.Value=5 (la propriété "value" de la cellule active reçoit la valeur 5)

  On notera qu'une variable peut recevoir sa propre valeur modifiée, par exemple lorsque l'on souhaite incrémenter un compteur de ligne : Numligne = Numligne + 1

- *Instructions exécutables (exécution d'une méthode, d'une fonction, structures de contrôle -si, alors, sinon - tant que-, affichage de boîtes de dialogue...)*

  MsgBox ("Veuillez taper la date sous la forme JJ/MM/AA") (affichage d'un message)

  Cells(Numligne, Numcolonne).Select (sélection de la cellule pour laquelle la valeur de la variable numligne détermine le numéro de ligne et numcolonne, le numéro de colonne)

3.2 Les deux principaux types de procédures

3.2.1 Procédure Sub

Elles exécutent un ensemble d'instructions sans renvoyer de valeur (par opposition aux procédures Functions). Par contre une procédure Sub peut recevoir des arguments de la part d'une procédure appelante.

```
Private Sub nom_proc(arguments, éventuels)
        Suite d'instructions
End, Sub
```

3.2.2 Procédure function

Une procédure function (fonction) renvoie une valeur qui est utilisée dans la procédure qui l'a appelée :

```
Private Function mafonc1 (arguments) as string
« as string » permet de qualifier le type de valeur que renverra la fonction (par ex string ou integer)
        'suivent des instructions
        ' et notamment celle-ci qui affecte une valeur à la fonction,
        'c'est cette valeur que renverra la fonction dans le programme qui l'utilise
        mafonc1=expression
End function
```

Les fonctions ainsi générées peuvent être utilisées dans Excel comme n'importe quelle autre fonction.

4 Mise en place de fonctions

Ouvrir le classeur "fonctions.xls"

La première feuille nommée TVA, propose de créer une fonction permettant de retrouver le montant hors-taxe à partir d'un montant TTC pour un taux de TVA donné.

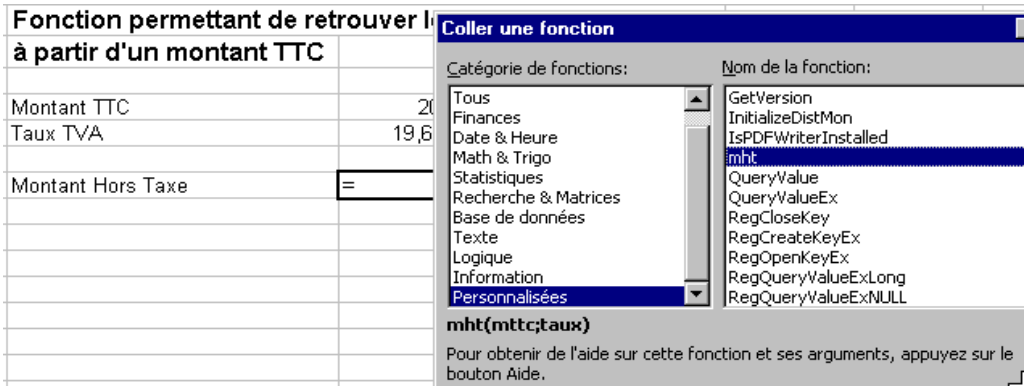
La fonction devra avoir comme argument "montant" et "taux"

1. Ouvrir VB Editor Outils/macro/VB Editor
   Ajouter un module

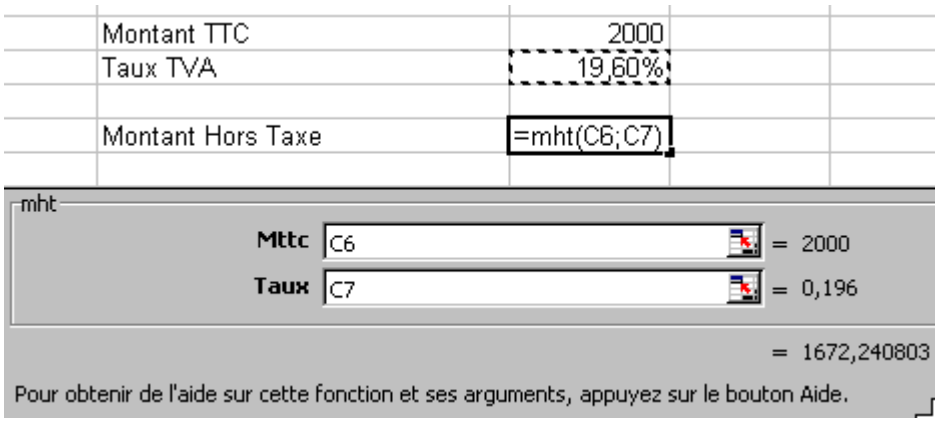
2. Saisir le code de la fonction

```
Function mht(mttc, taux)
    mht = (mttc / (1 + taux))
End Function
```

3. Revenir dans Excel et insérer cette fonction personnalisée dans la cellule C9

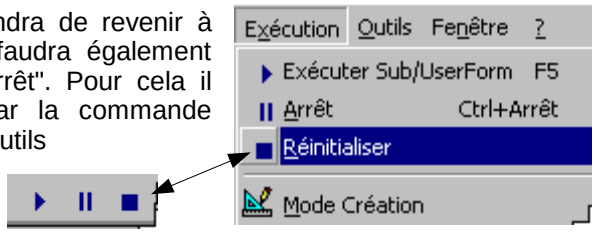


4. Renseigner les paramètres de la fonction puis OK pour voir le résultat renvoyé par cette fonction.



Remarque :

En cas d'erreur dans le code de la fonction, il conviendra de revenir à l'éditeur de Visual Basic, de corriger l'erreur, mais il faudra également réinitialiser la fonction, car l'exécution est en mode "arrêt". Pour cela il convient d'activer la touche de réinitialisation soit par la commande réinitialiser du menu outil, soit par le bouton de la barre d'outils



Cette fonction est disponible dans la feuille « TVA » du classeur « fonctions_cor.xls »

Bien entendu pour un tel calcul, il n'est pas vraiment nécessaire d'utiliser une fonction, mais dans certains cas plus complexes, le recours à une fonction permet de simplifier l'activité de l'utilisateur...

Ainsi par exemple on peut générer une fonction qui renvoie la longueur de l'arc d'une hélice... Il y a peu de chances que cette fonction existe en standard dans Excel....
Cf. feuille « hélice » du classeur "Fonctions.xls"

Les équations paramétriques de l'hélice sont :

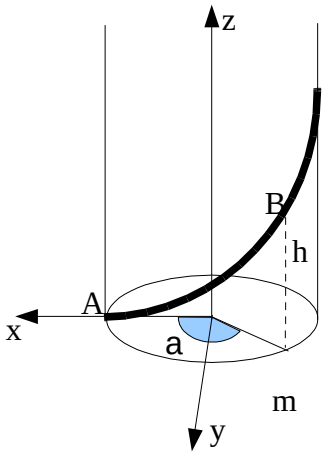
x=R cos a

y=R sin a

z = b a où b=h / 2 (b est le pas réduit de l'hélice)¶

La longueur de l'arc de l'hélice est

$$s= \quad a\sqrt{R^2+\frac{h^2}{(4¶^2)}}$$



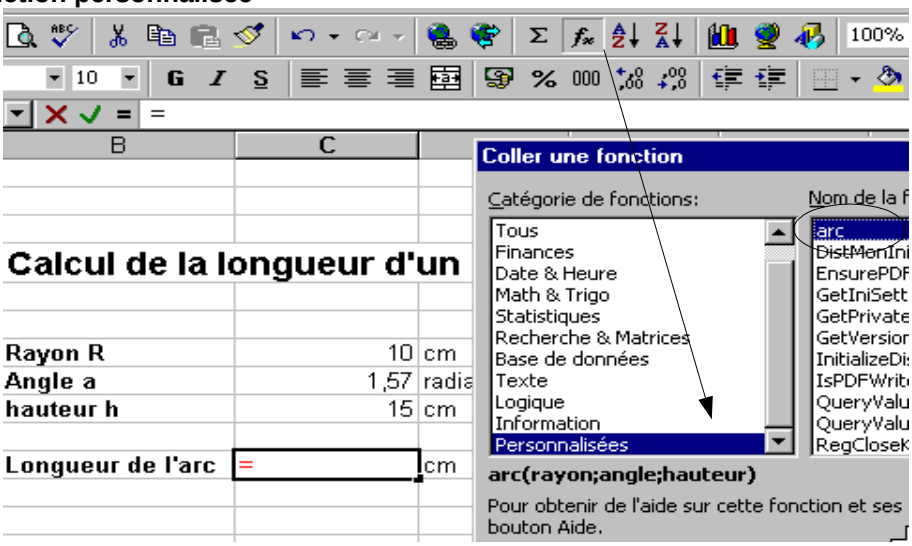
La fonction devra recevoir pour paramètres

- le rayon R du cylindre,
- l'angle en radian,
- la hauteur h

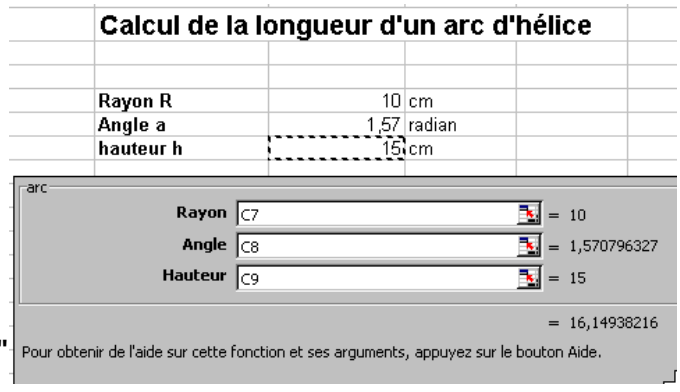
Saisie de la fonction dans un nouveau module sous VB Editor

```
Function arc(rayon, angle, hauteur)
  arc = angle * Sqr((rayon ^ 2) + ((hauteur ^ 2) / (4 * (3.1416 ^ 2))))
End Function
```

Appel de la fonction personnalisée



Saisie des paramètres



Le corrigé se trouve dans la feuille « hélice » du classeur "Fonctions_cor.xls"

Table des matières