

# Introduction au langage VBA



## Table des matières

1.	Introduction .....	1
2.	Mise en place .....	1
3.	Première application .....	5
I.	Premier exemple .....	5
II.	Deuxième exemple. ....	12
III.	Troisième exemple. ....	13
IV.	Quatrième exemple .....	14
4.	Le modèle Objet d'Excel .....	15
I.	Quelques méthodes de l'objet <i>Workbook</i> .....	15
II.	Quelques méthodes de l'objet <i>Worksheet</i> .....	16
III.	Les variables .....	16
IV.	Les structures de décision et de répétition .....	18
i.	Le If .....	18
ii.	Le Select Case .....	19
iii.	La structure de répétition For Next .....	20
iv.	La structure de répétition <i>For Each</i> .....	20
v.	La boucle Do While et Do Until .....	20
5.	Les formulaires .....	21
6.	Exemples de codes .....	25
I.	Problèmes .....	25
i.	Exemple Base de données- Liste des enseignants de l'institut Teccart .....	25
ii.	Exemple Population du Canada .....	25
II.	Les codes .....	27
i.	Exemple Base de données .....	27
ii.	Exemple Population du Canada .....	29

## 1. Introduction

Le langage VBA (*Visual Basic pour Applications*) est un langage de programmation permettant d'interagir avec les applications de la suite *Office* de *Microsoft*. Le VBA est une déclinaison du VB (c'est en quelque sorte, son petit frère). C'est un langage pouvant tourner sur les différentes applications de la suite office chacune avec ses propres spécificités. Autrement dit, au moyen de ce langage, on est en mesure de programmer des tâches se déroulant à l'intérieur des applications *Office*, et, parce que chaque application a ses propres commandes, Il existe donc, une version VBA pour chaque application avec un tronc commun, certaines actions étant communes à toutes les versions. La figure<sup>1</sup> suivante illustre cette situation pour la suite Office 2010.



Figure 1

## 2. Mise en place

Dans ce cours, on s'intéressera à la version VBA pour Excel parce que c'est la version où la programmation prend tout son sens (manipulation de chiffres, formules,...). Donc, si on démarre *Excel* (ici, version 2013), on verra un *Ruban* comme celui de la figure 2, et après un clic sur l'onglet (ou menu) **Fichier**, on choisit le menu **Options, Personnaliser le ruban**. La fenêtre (figure 3) se présente et vous devez cocher la case **Développeur**. Le ruban affichera dorénavant l'onglet **Développeur** (figure 4). (Pour la version 2007, bouton **Office**, Menu **Options Excel**, option **Standard** et cocher **Afficher l'onglet Développeur dans le ruban**).

<sup>1</sup> [WWW.lesprosdelasouris.fr](http://WWW.lesprosdelasouris.fr)

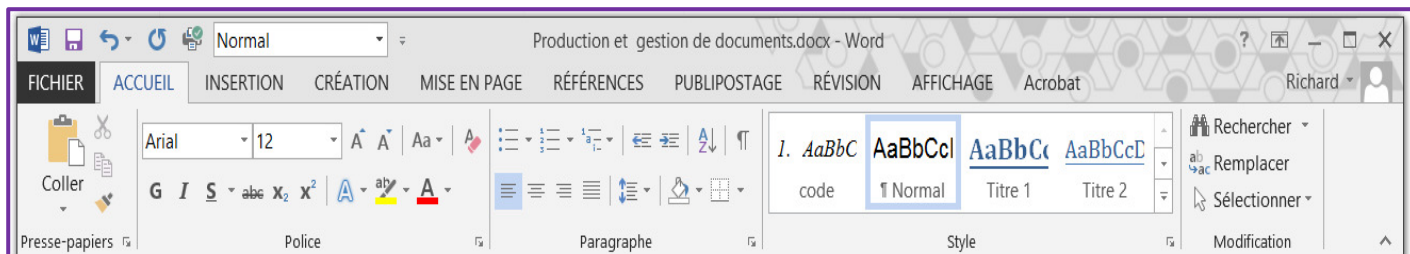


Figure 2

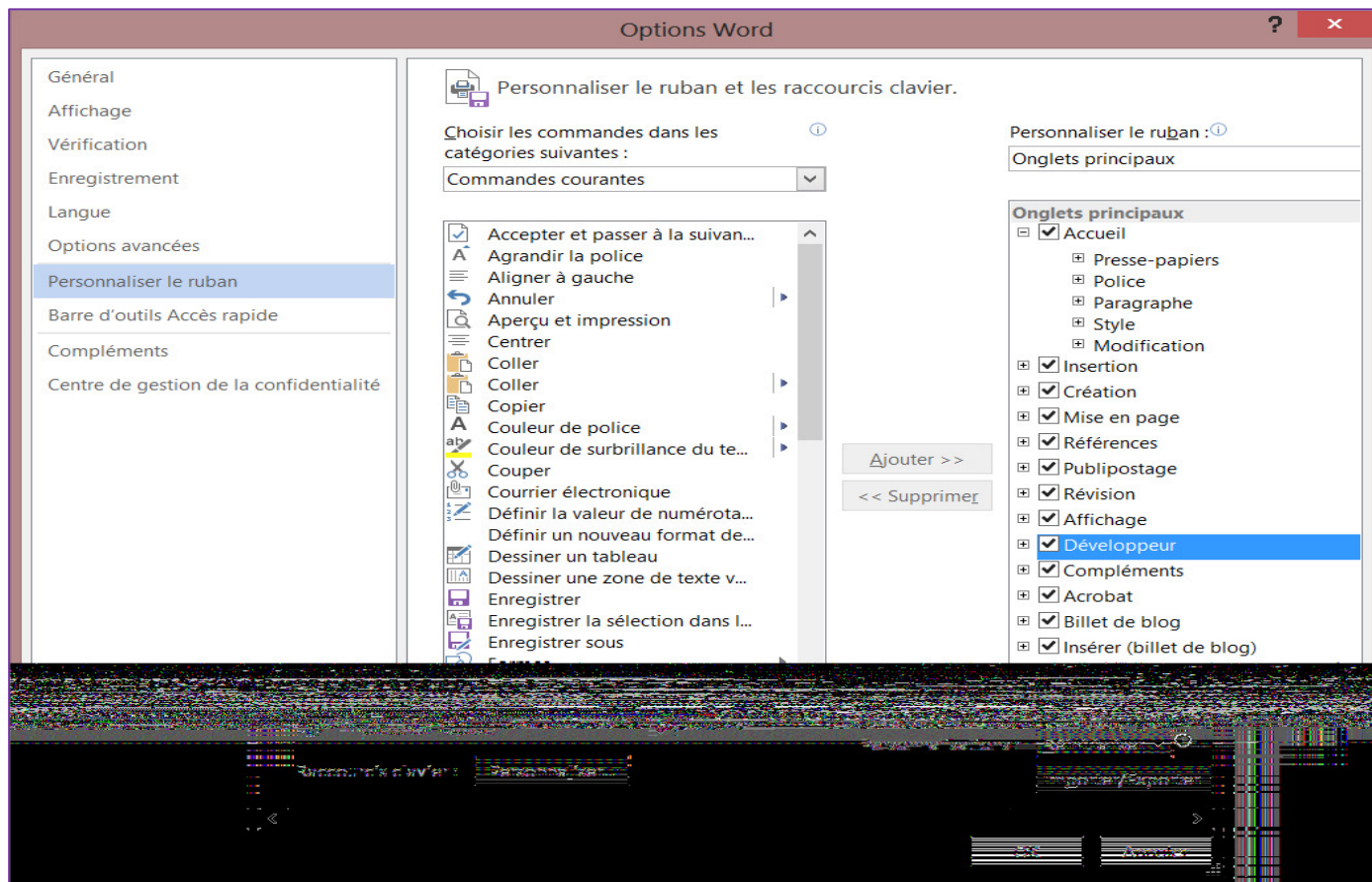


Figure 3

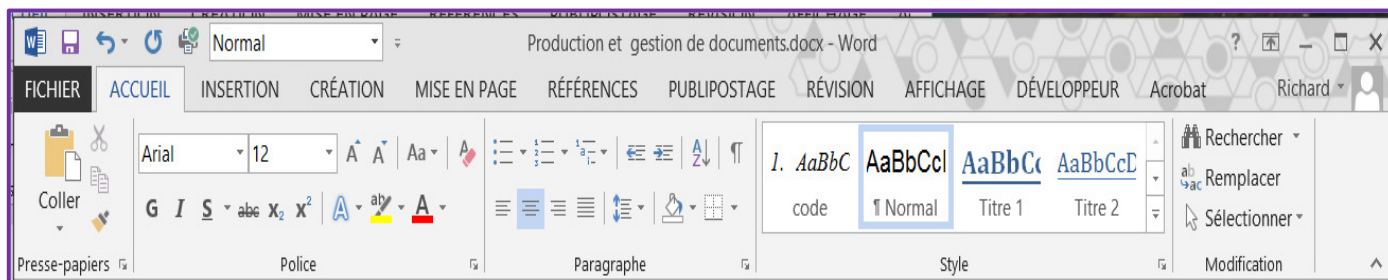
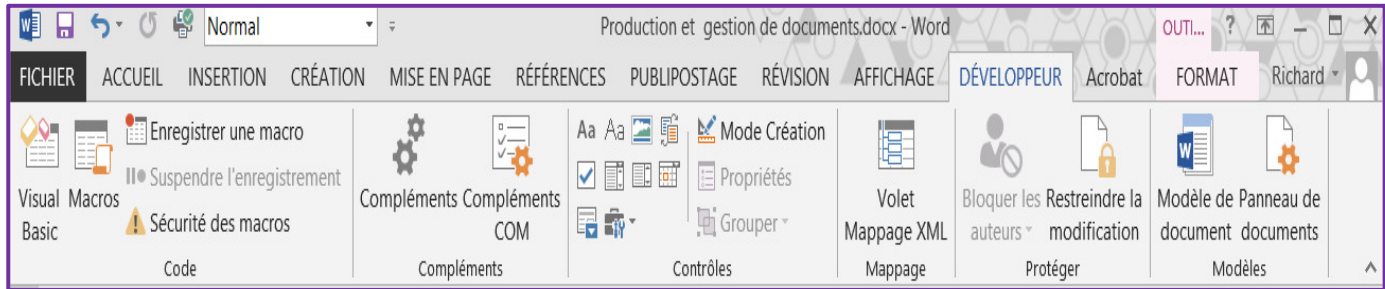


Figure 4



Si on clique sur cet onglet, le Ruban (légèrement différent de la version 2007) prendra alors la forme suivante :

Figure 5

On remarque quelques nouvelles zones (ou groupes) : **code**, **Compléments** et **Contrôles**.

Office 2013 est livré avec un niveau de sécurité assez élevé de telle sorte qu'il refusera même d'exécuter nos programmes VBA. Pour corriger cette situation, nous devons nous déplacer vers le menu Centre de gestion de la confidentialité du menu **Options** (que l'on atteint au moyen de l'onglet **Fichier** (bouton *Office* pour la version 2007)). On clique ensuite sur le bouton Paramètres du Centre de gestion de la confidentialité... pour voir la figure 6 suivante.

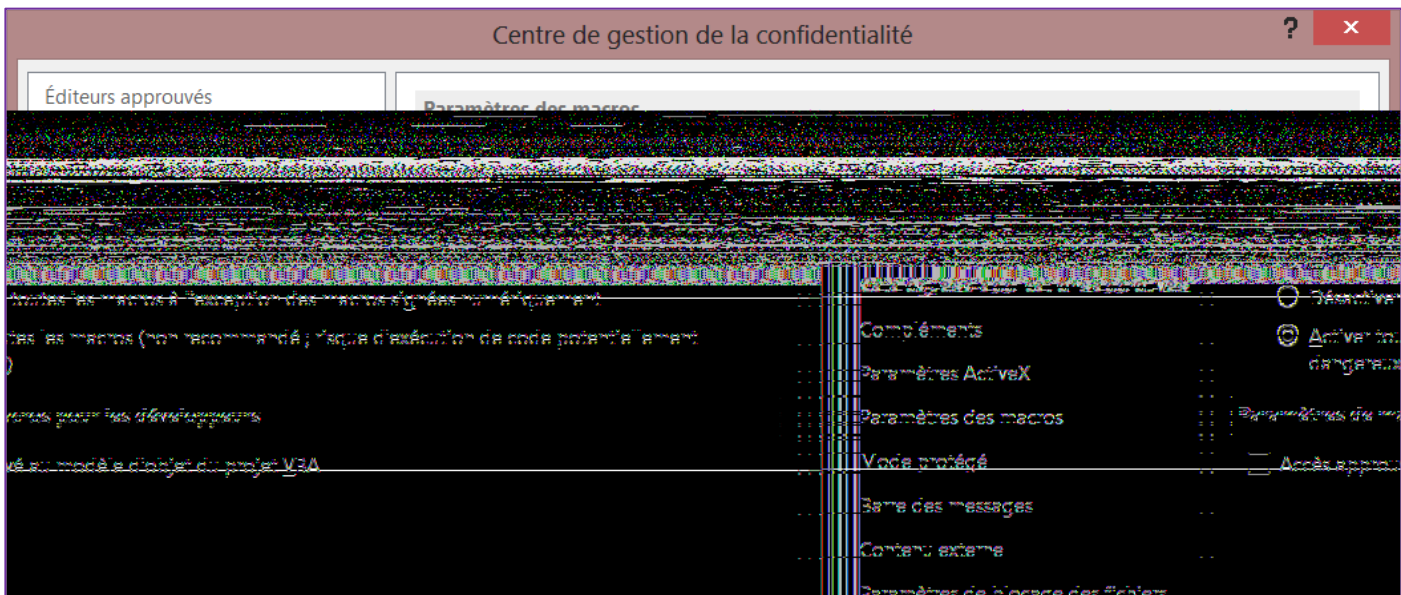


Figure 6

Nous allons ici choisir Activer toutes les macros malgré les avertissements. N'oubliez-pas de remettre cette option au choix Désactiver toutes les macros avec notification si vous avez à ouvrir des classeurs contenant des macros en provenance de l'extérieur.

Finalement, comme nous allons créer des projets VBA, il faut que l'on enregistre nos classeurs *Excel* dans un nouveau format, plus sécurisé. Si on procède à un enregistrement de classeurs suivant le mode habituel, le message suivant (figure 7) se présentera :

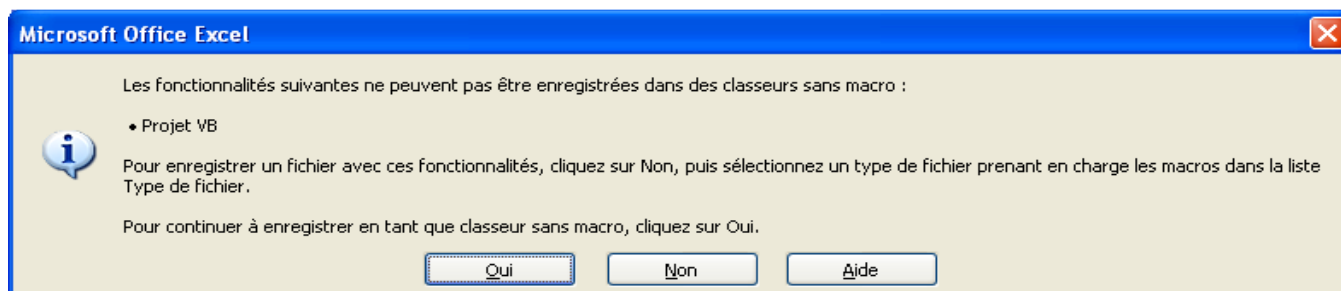


Figure 7

Il faudra alors enregistrer nos classeurs contenant du VBA ou des macros sous le type Classeur Excel prenant en charge les macros :

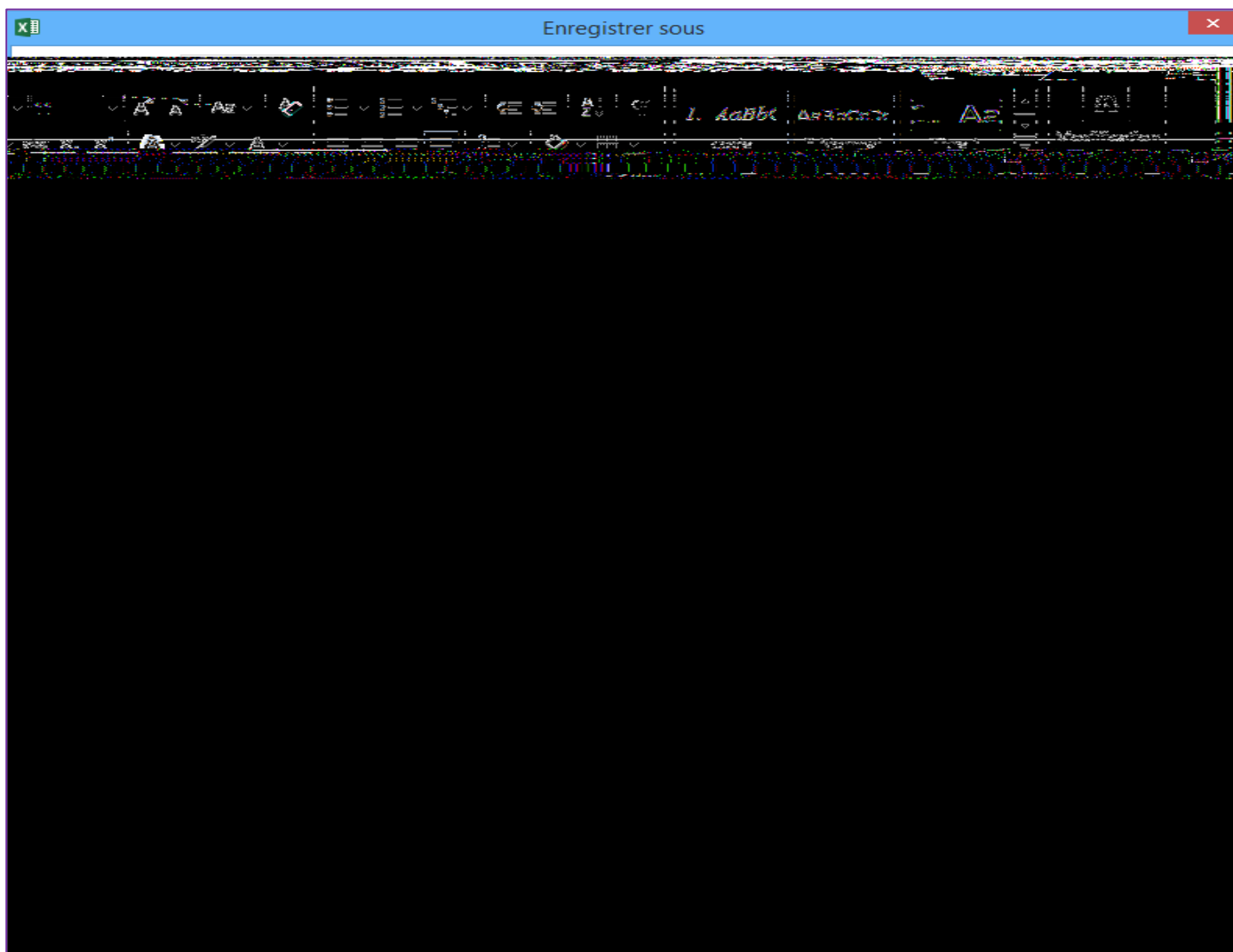


Figure 8

Prenez note que l'extension des fichiers *Excel* qui est normalement **.xlsx** deviendra alors pour ce type de fichiers **.xlsm**.

### 3. Première application

Comme première application, nous allons créer un classeur *Excel* que nous allons enregistrer sous le nom *exemple1.xlsm* dont le contenu est le suivant:



Figure 9

#### I. Premier exemple

Nous allons faire afficher le traditionnel message *Bonjour au monde du VBA!*

Pour débiter, nous allons cliquer sur l'onglet **Développeur** du *Ruban* et ensuite sur l'icône Mode de création. Dans ce mode, nous pourrions placer des boutons sur la feuille de calcul, les déplacer, écrire du code...Mais pour être en mesure d'exécuter le code que nous aurons écrit, il faudra quitter le mode de création.

On clique sur cet icône et ensuite sur l'icône Insérer: On verra la figure suivante (10) se présenter. On peut y remarquer deux zones contenant chacune des *contrôles*: la zone **formulaire** et la zone **ActiveX**. Les contrôles sont des objets qui pourront se retrouver sur notre feuille de calcul *Excel* comme des boutons de commande, des zones de listes déroulantes, des cases à cocher, des zones de listes, des boutons d'options, des zones de texte, des barres de défilement, des toupies, des étiquettes....



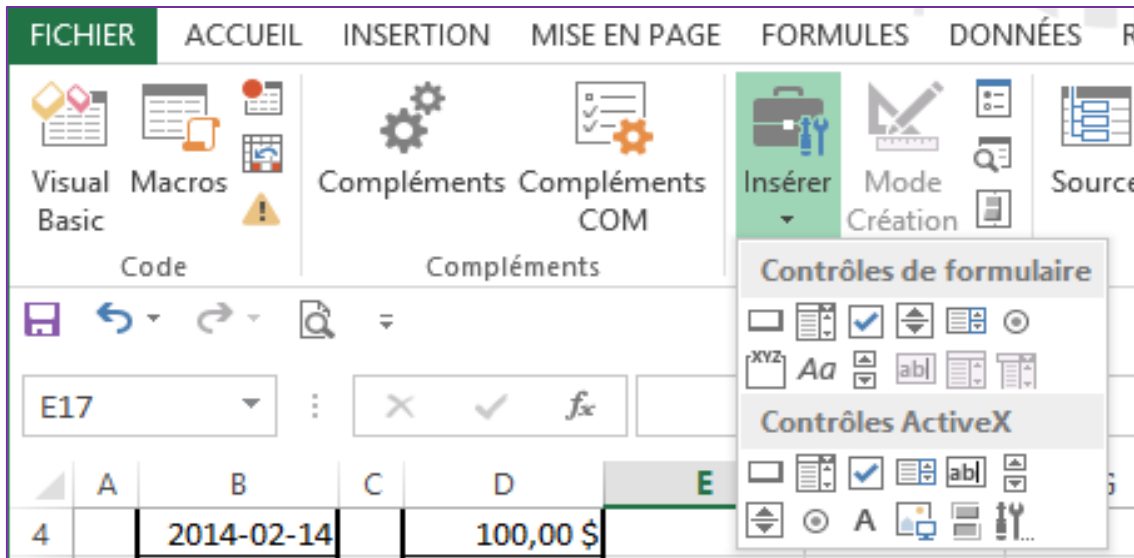


Figure 10

Nous n'allons d'abord utiliser que les contrôles de type *ActiveX*. Les contrôles de type *Formulaires* nous permettraient d'enregistrer des formulaires sous la forme de macros commandes. Voici une définition d'un contrôle *Activex*<sup>2</sup>:

*Un contrôle ActiveX peut être une simple zone de texte ou peut se révéler plus complexe, comme une barre d'outils spéciale, une boîte de dialogue ou une petite application. Les contrôles ActiveX sont utilisés sur les sites Web ou dans les applications de votre ordinateur. Les contrôles ActiveX ne sont pas des solutions autonomes. Ils ne peuvent être exécutés que dans des programmes hôtes comme Windows, Internet Explorer et les programmes Microsoft Office. Les contrôles ActiveX sont toutefois très puissants car, en tant qu'**objets COM (Component Object Model : spécification développée par Microsoft pour créer des composants logiciels pouvant être assemblés en programmes ou être utilisés pour ajouter des fonctionnalités à des programmes s'exécutant sur le système d'exploitation Microsoft Windows.)**, ils ont un accès illimité à votre ordinateur. Les contrôles ActiveX peuvent accéder au système de fichiers local et modifier les paramètres du Registre du système d'exploitation. Si un pirate utilise un contrôle ActiveX pour prendre le contrôle de votre ordinateur, les dommages peuvent être considérables.*

Ce sont donc des outils puissants qu'il faudra utiliser avec prudence... On choisira le bouton de commande de la zone *Activex*. C'est le premier bouton de cette zone (voir figure 11).

On clique une fois sur le contrôle voulu, le curseur prend alors la forme d'une croix et on peut ensuite se placer à l'endroit voulu sur la feuille de calcul. Il suffit ensuite de cliquer en maintenant le bouton enfoncé et de choisir la grandeur voulue du bouton. On obtiendra alors une feuille de calcul analogue à la figure 12.

<sup>2</sup> Aide de la suite Office 2007

On remarque des points entourant le bouton. Voyons maintenant comment en changer le texte et l'apparence. Pour ce faire, on effectue un clic droit sur le bouton et on choisit le menu propriétés du menu contextuel ou encore on clique sur le menu propriétés du Ruban dans la zone *contrôles*:

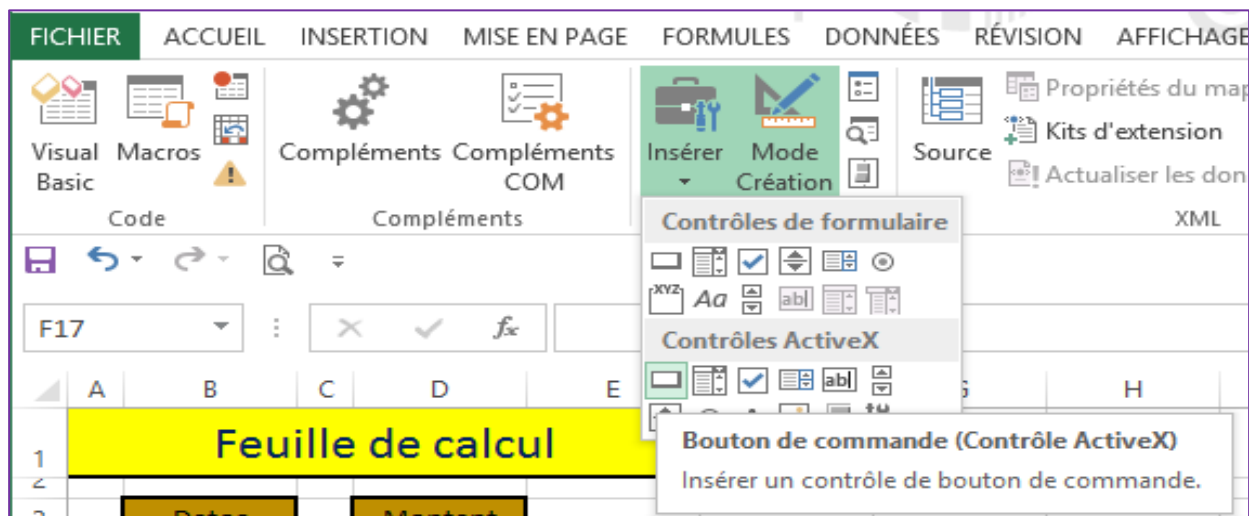


Figure 11

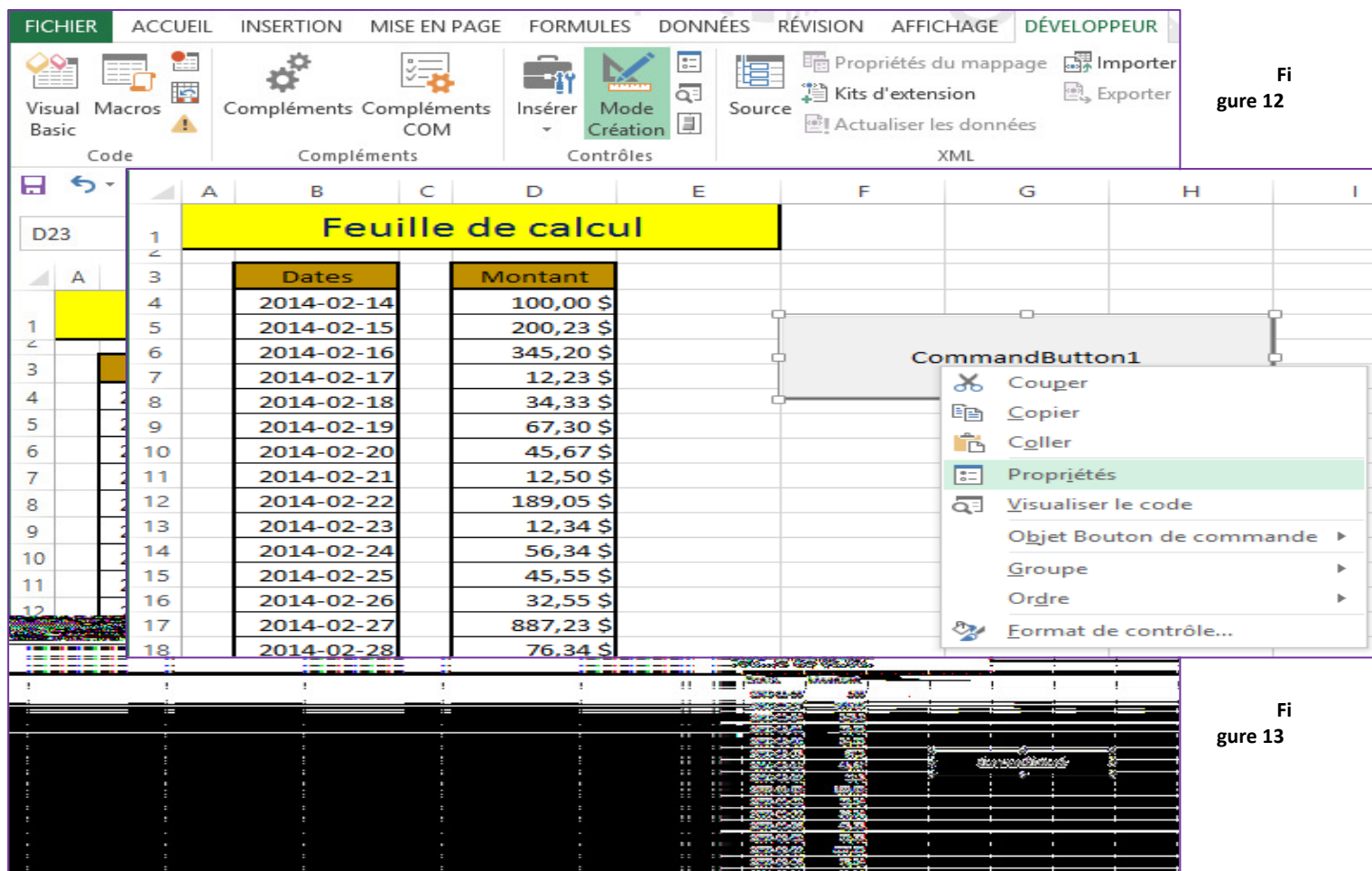


Figure 12

Figure 13

On verra alors la fenêtre suivante se présenter la figure suivante (14):

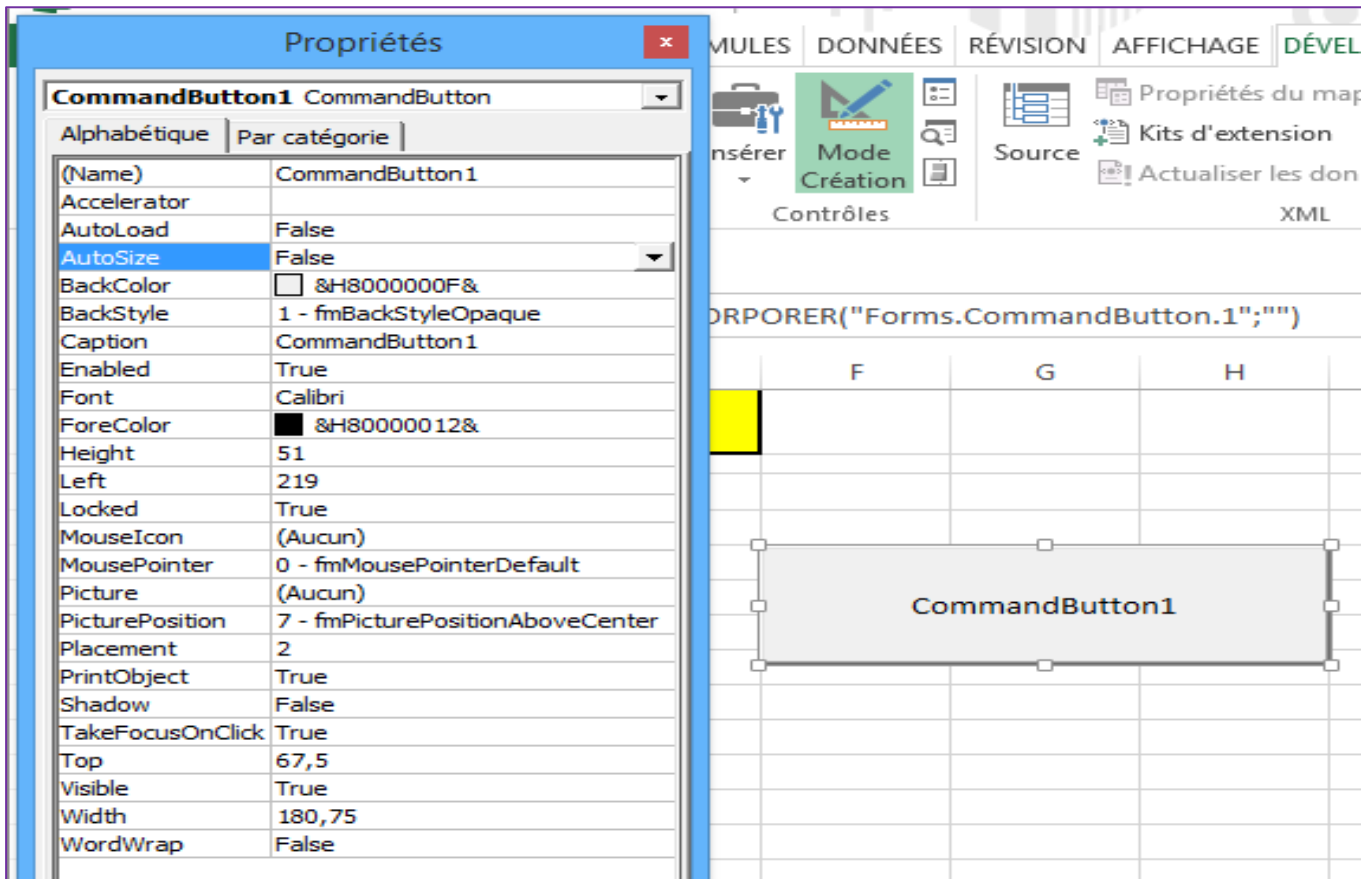


Figure 14

présenter quelques-unes.

Tout d'abord, chaque contrôle, que l'on place sur la feuille, se voit attribuer un nom par défaut par VBA. Ici, le nom (name) est *CommandButton1*. Nous allons le changer pour *butAfficheMessage*. Vous aurez à faire de même lors de l'insertion de contrôles sur vos feuilles de calculs.

Pour les boutons de commande, la propriété *Caption* contient le texte qui est affiché. Nous allons le changer pour *Cliquer s.v.p.*

Les propriétés *BackColor* (couleur du fond) *Font* (Police) et *ForeColor* (couleur de la police) se passent de commentaires.

La propriété *Enabled* est fixée à *True* (Vrai). Cela rend le bouton de commande disponible. Si on double clique sur la valeur *True*, VBA changera le *True* pour un *False*. Dans ce cas, le bouton apparaît en grisé sur la feuille et n'est pas fonctionnel.

La dernière propriété intéressante pour nous est *Visible*. Encore ici, son utilisation est assez évidente : une valeur *True* rend le contrôle visible alors qu'une valeur *False* le rend invisible.

On verra comment modifier ces propriétés lors de l'exécution du code de nos programmes VBA.

Changeons maintenant le nom et le texte affiché de notre bouton de commande. On verra la figure suivante :

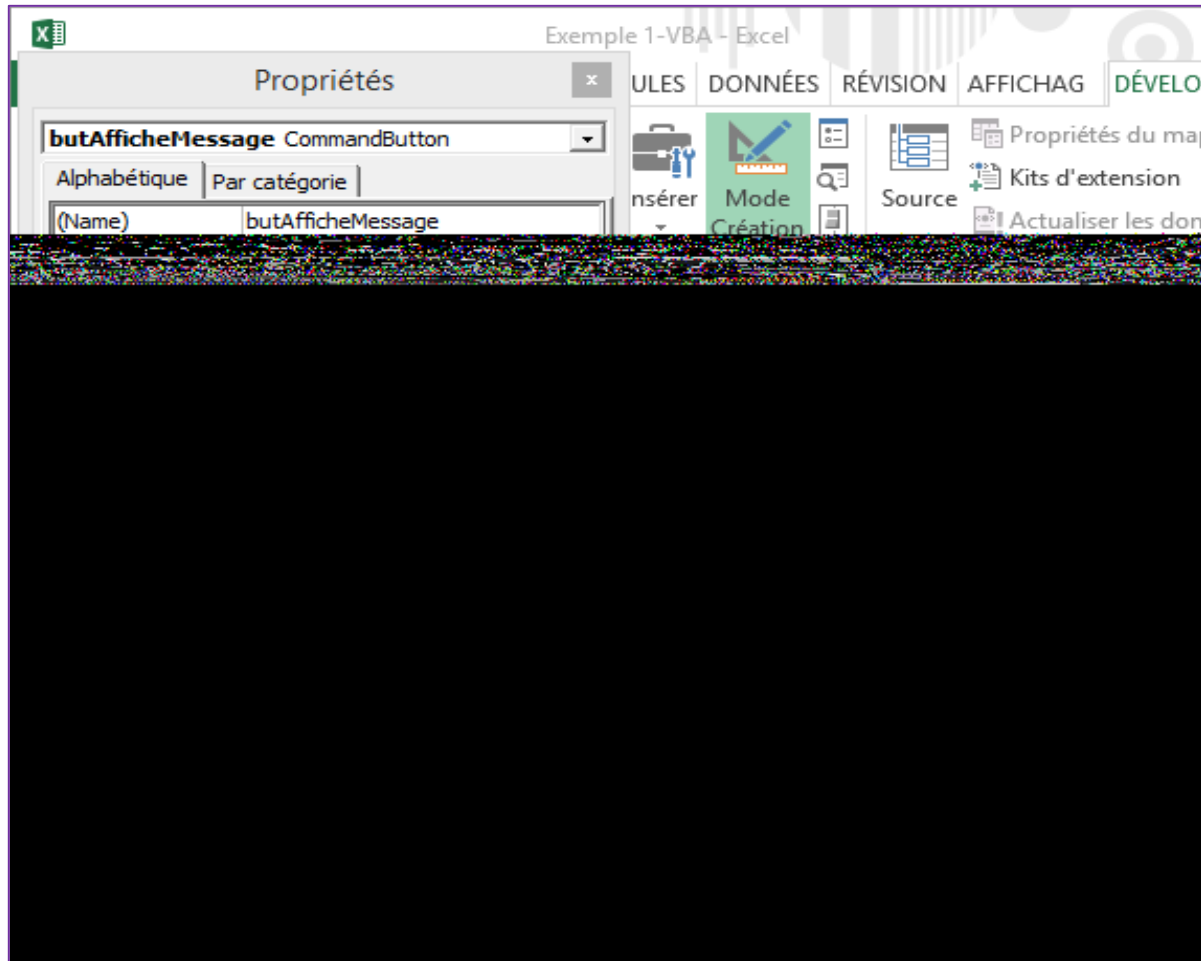
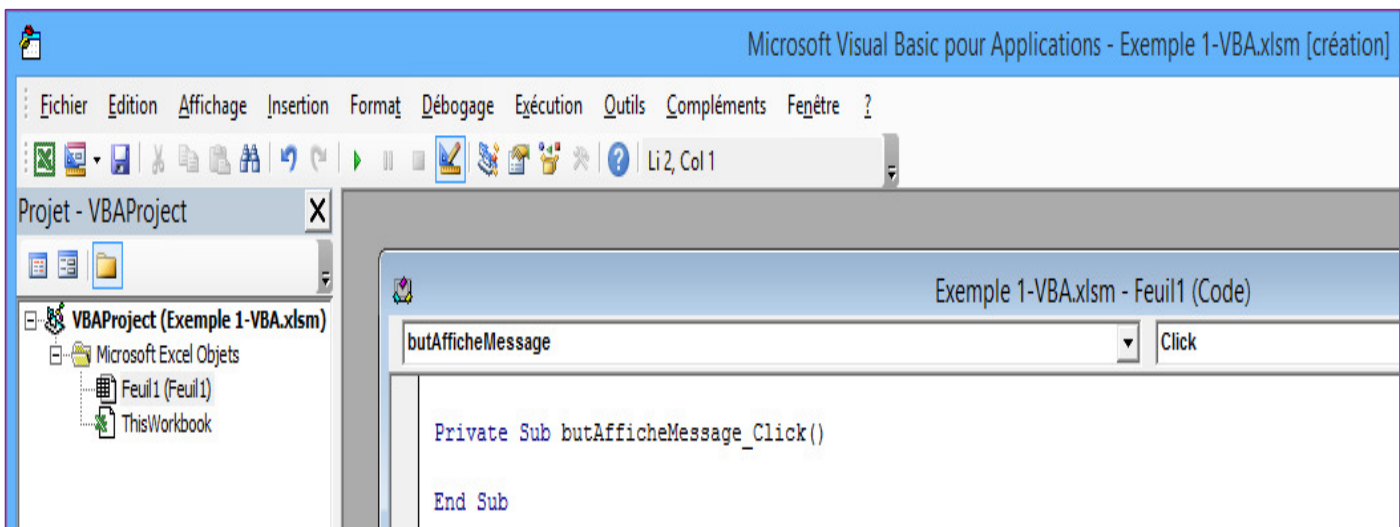


Figure 15

Voyons maintenant comment placer du code derrière ce bouton : on peut soit double cliquer sur le bouton ou encore cliquer sur l'icône *Visual Basic* se trouvant tout à gauche du *Ruban* de l'onglet *Développeur*. Dans les deux cas, la fenêtre suivante se présente :





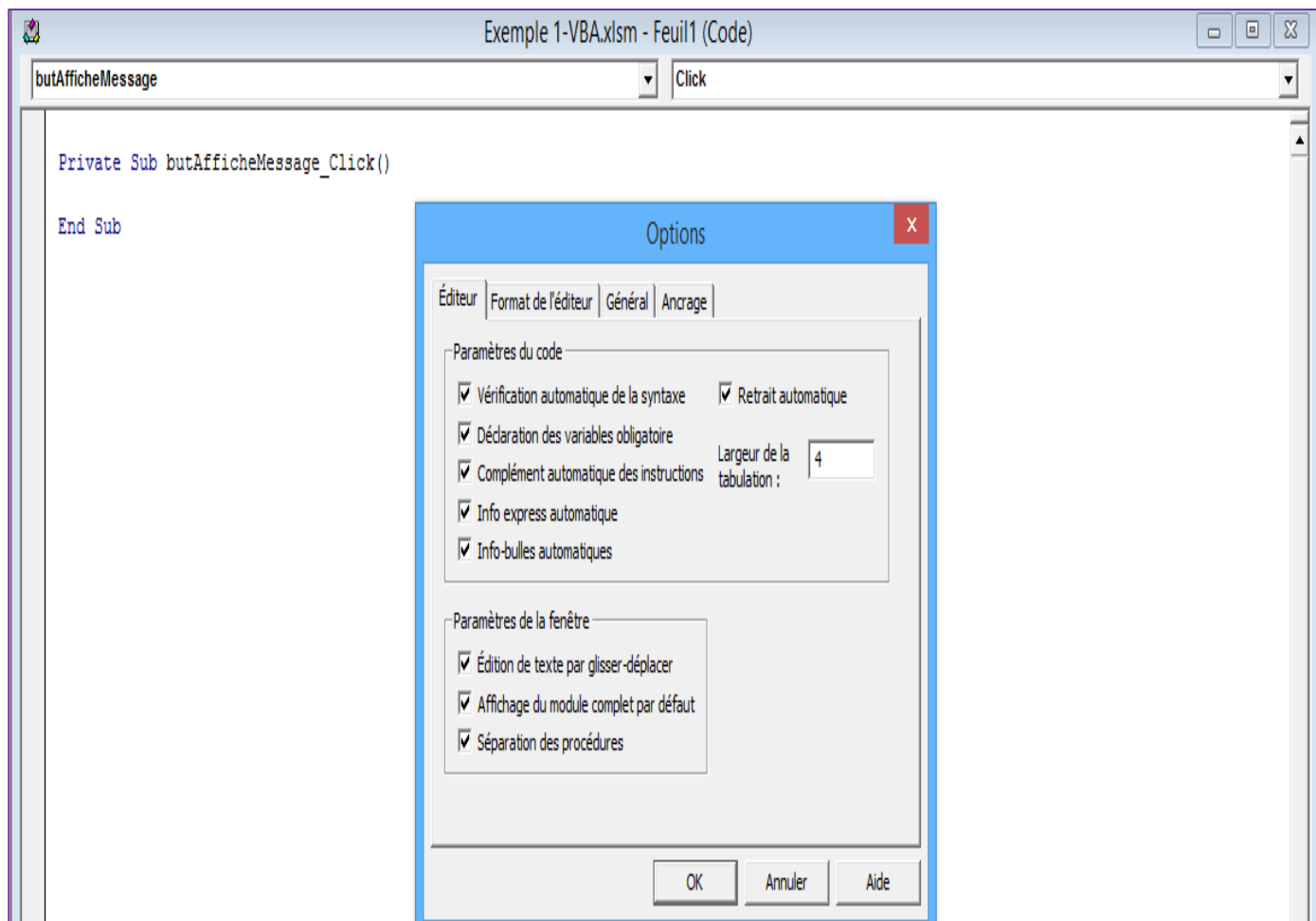
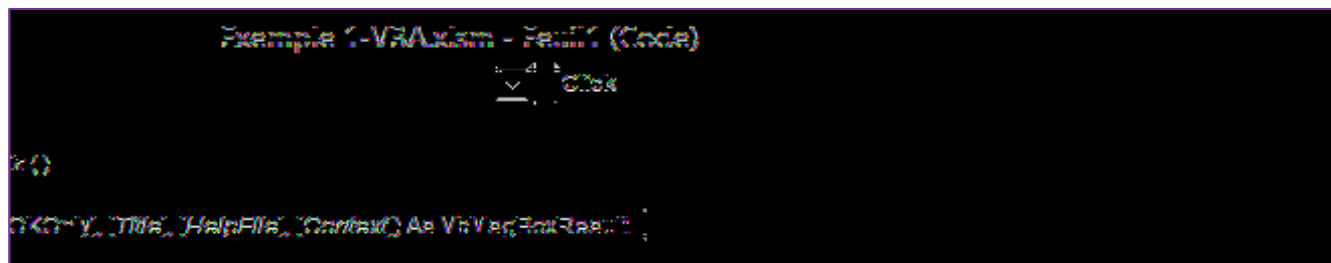


Figure 18

Pour programmer notre bouton , on fera appel à une fonction VBA nommée **MsgBox()** qui nous permettra d'afficher le message de notre choix. On place le curseur entre **Private Sub butAfficheMessage\_Click()** et **End Sub**. On entre ensuite **msgbox(** , on voit alors un message d'aide apparaître, tel qu'illustré à la figure suivante (19):



On peut remarquer que l'éditeur VBA a réécrit le mot `msgbox` comme `MsgBox` en y plaçant des lettres majuscules : Le langage VBA n'est pas sensible à la casse, il la gère. On entre ici simplement un message que l'on doit placer entre guillemets (remarquez l'absence de ; à la fin de la ligne. Ce n'est pas nécessaire en VBA) :

**MsgBox("Bienvenue au monde du VBA")**

Pour exécuter ce programme, on peut réduire la fenêtre *Microsoft Visual Basic* dans laquelle nous venons d'introduire le code, ramener à l'avant plan la feuille de calcul et son bouton. Si la feuille de calcul se trouve toujours en mode Création, il ne nous sera pas possible d'exécuter ce programme. Dans ce cas, il suffit de cliquer à nouveau sur le bouton *Mode Création*. On clique ensuite sur le bouton pour voir apparaître la fenêtre suivante :

Feuille de calcul		
Dates	Montant	
2014-02-14	100,00 \$	
2014-02-15	200,23 \$	
2014-02-16	345,20 \$	
2014-02-17	12,23 \$	
2014-02-18	34,33 \$	
2014-02-19	67,30 \$	
2014-02-20	45,67 \$	
2014-02-21	12,50 \$	
2014-02-22	189,05 \$	
2014-02-23	12,34 \$	
2014-02-24	56,34 \$	
2014-02-25	45,55 \$	
2014-02-26	32,55 \$	
2014-02-27	887,23 \$	
2014-02-28	76,34 \$	
2014-03-01	87,34 \$	
2014-03-02	22,33 \$	
2014-03-03	22,56 \$	

Cliquer S.V.P.

Microsoft Excel  
Bienvenue au monde du VBA  
OK

Figure 20

## II. Deuxième exemple.

Voici le code du bouton nommé **butExempleDeif()** qui s'applique à la feuille de calcul précédente. Un clic sur ce bouton va colorer en bleu toutes les cellules d'une plage dont la valeur est supérieure à 300. En voici le code

```

1. Private Sub butExempleDeIf_Click()
2.     Dim maPlage As Range
3.     Dim Montant As Range
4.     Set maPlage = Range("D4:D21")
5.     ' début de la boucle
6.     For Each Montant In maPlage
7.         ' On vérifie le montant
8.         If (Montant.Value > 300) Then
9.             ' les 56 couleurs sont disponibles sur http://www.mvps.org/dmccritchie/excel/colors.htm
10.            Montant.Interior.ColorIndex = 22
11.        End If
12.    Next
13. End Sub

```

Les lignes 2 et 3 définissent les variables **maPlage** et **Montant** comme des variables de type **Range**, pouvant contenir une ou plusieurs cellules. Remarquez l'instruction **Dim** qui sert à définir le type d'une variable en VBA.

La ligne 4 attribue à la variable **maPlage** les cellules comprises entre la cellule D4 et la cellule D21.

La ligne 6 va de pair avec la ligne 12. On a ici une structure de répétition, une boucle **For Each** (pour chaque valeur de la plage...) qui exécutera les lignes 8, 10 et 11 pour chaque cellule de la plage.

La ligne 6 place successivement les valeurs de la plage dans la variable **Montant**. Par la suite, on utilise une structure de décision, un **if**, dont voici la structure générale :

```

If (condition vrai) then (alors)
    fait une action
else(sinon)
    fait autre chose
End if (fin de la structure de décision)

```

À la ligne 8, on vérifie si la valeur de la plage actuellement analysée est plus grande que 300. Si oui, la ligne 10 lui applique la couleur de fond numéro 22 qui est un rose foncé. Si la condition est fausse, aucune action n'est entreprise ici.

### III.Troisième exemple.

Dans cet exemple, on utilise l'objet **Application.InputBox()** afin de demander à l'utilisateur de choisir une cellule dont le contenu sera multiplié par 2 et affiché dans une autre cellule.

```

1. Private Sub cmdFoisDeux_Click()
2.     Dim cellAffiche As Range, laSelection As Range
3.     Set cellAffiche = Range("M12")
4.     cellAffiche.Value = ""
5.     Set laSelection = Application.InputBox("Sélectionner une cellule", Type:=8)
6.     If laSelection Is Nothing Then
7.         MsgBox ("Faites une sélection")
8.     Else
9.         cellAffiche.Value = laSelection.Value * 2
10.    End If
11. End Sub

```

La ligne 4 utilise la propriété **value** d'une cellule afin de la vider au moyen des deux guillemets collés. La ligne 5 demande à l'utilisateur de sélectionner une cellule, une plage en fait. Le **type := 8** s'assure que l'utilisateur entre bien une plage. Si on avait utilisé le type :=1, VBA se serait assuré que l'utilisateur a bien choisi un nombre, pour type :=2, il aurait attendu une chaîne de caractères. La ligne 6 vérifie qu'une sélection a bien été faite et si oui, calcule à la ligne 9 le produit par 2 et l'affiche dans la cellule **cellAffiche**.



## IV. Quatrième exemple

Dans ce cas, l'exemple est plus complexe. Nous avons placé sur la feuille deux étiquettes, deux boîtes de texte et un bouton de commande. Ce programme demande à l'utilisateur d'entrer le nombre de lignes qu'il désire additionner dans la colonne des chiffres et au clic du bouton, il affiche la réponse dans un contrôle boîte de texte. Voici le code du bouton :

```
1. Private Sub cmdAdditionne_Click()  
2.     Dim nb, i As Integer, total As Double, valeur As Range, maPlage As Range  
3.     Set maPlage = Range("D4:D21")  
4.     nb = Val(txtNombre.Text)  
5.     total = 0  
6.     For i = 1 To nb  
7.         total = maPlage(i) + total  
8.     Next  
9.     txtRep.Text = total  
10. End Sub
```

La ligne 2 définit des variables de type **Integer** (*entier*), de type **Double** (point flottant) et de type **Range**. La ligne 4 va chercher la valeur du texte se trouvant dans la boîte de texte nommée **txtNombre**. Pour ce faire, on fait appel à sa propriété **Text**. Ainsi, **txtNombre.Text** retourne le texte se trouvant dans la boîte de texte. Mais comme c'est du texte, la fonction **Val()** le convertit en nombre.

La ligne 6 débute une autre forme de la boucle for :

```
For variable = debut To fin  
    Lignes à répéter  
    ...  
Next variable
```

Dans cette forme, on utilise une variable de contrôle (*ici i*) qui se voit attribuer une valeur minimum et une valeur maximum. Suivent ensuite les lignes devant être répétées et le tout se termine par l'instruction **Next** qui peut être suivie du nom de la variable de contrôle. Donc, dans notre exemple, c'est l'utilisateur qui détermine le nombre de répétitions à exécuter.

La ligne 7 cumule les différentes valeurs de la plage. Remarquez comment on peut accéder aux différentes cellules d'une plage. On utilise un index, comme dans **maPlage(index)** où index peut prendre les valeurs de 1 à la grandeur de la plage. Dans notre exemple, **maPlage(1)** correspond à la valeur 100, **maPlage(2)** à la valeur 200,23 etc.

La ligne 9 affiche le résultat dans la boîte de texte **txtrep.Text**. Nous illustrerons le tout au moyen d'un exemple en classe.

## 4. Le modèle Objet d'Excel

Dans cette section nous présenterons les éléments de base permettant de programmer en VBA. Nous présenterons d'abord le modèle objet d'Excel pour enchaîner avec les éléments de programmation plus standards.

La programmation VBA suppose qu'Excel est un objet contenant plusieurs autres objets. Voici une liste hiérarchisée de certains de ces objets qui nous intéresseront :

- *Application* (en fait, c'est Excel lui-même)
  - *Workbook* (un classeur)
    - *Worksheet* (une feuille de calcul Excel)
      - *Range*
    - *Chart* (un graphique d'Excel dans sa propre feuille)
    - *VBProject* (un projet VB)

Veillez noter que cette liste d'objets n'est pas exhaustive. Excel comporte de plus des *collections* qui sont en fait des structures analogues à des tableaux et qui contiennent des objets précis. Voici quelques collections qui nous seront utiles :

*Workbooks*, *Worksheets*, *Charts* et *Sheets*.

Rappelons qu'Excel peut contenir plusieurs *Workbook* (classeurs). De plus, un classeur Excel peut contenir plusieurs *Worksheets* (feuilles de calcul) et/ou plusieurs *Charts*.

Pour faire référence à un de ces objets, on peut procéder de deux façons. Par exemple, si on veut rendre active la feuille de calcul numéro 3 qui se nomme *Étape\_2* (avant d'y effectuer une opération...) du classeur 1, on pourrait écrire :

```
Application.Workbooks(1).Worksheets("Étape_2").Activate
```

ou

```
Application.Workbooks(1).Worksheets(3).Activate
```

On peut donc utiliser le nom de la feuille de calcul ou son numéro. Les numéros commencent à 1. Remarquez de plus la présence d'une feuille contenant seulement un graphique, une feuille de type *graphique* ou *Chart* dont le nom est *Graphique1*. Ces feuilles sont numérotées elles aussi à partir de 1.

Même s'il est possible de négliger les termes *Application.WorkBook(1)* lorsque notre code VBA se trouve dans le même classeur que la feuille (*Worksheet*) référencée, on recommande néanmoins de les maintenir.

### I. Quelques méthodes de l'objet *Workbook*

**Activate** : active le classeur

**Close** : ferme le classeur

## II. Quelques méthodes de l'objet *Worksheet*

VBA nous permet d'ajouter de nouvelles feuilles de calcul, de supprimer des feuilles de calcul, d'en copier le contenu.... En voici quelques-unes

**Add** : permet d'ajouter une feuille à un classeur

**Activate** : active la feuille

**Calculate** : force une mise à jour de la feuille

**Copy** : copie la feuille.

Ex : Application.Workbooks(1).Worksheets(1).Copy After : = Worksheets("Étape\_3") ,  
copierait la première feuille de calcul vers une feuille de calcul se trouvant après la feuille  
nommée *Étape\_3*.

**Delete** : supprime la feuille de calcul référencée.

Ex : Workbooks(1).Worksheets("Étape\_3").Delete

**Move** : déplace une feuille de calcul

**Paste** : colle une feuille de calcul

Notez que plusieurs de ces fonctions s'appliquent aussi à des plages de cellules.

## III. Les variables

Une variable est le nom que l'on donne à une case mémoire devant contenir des données. Une variable est caractérisée par son nom, son type et sa portée.

En VBA, le nom des variables doit commencer par une lettre et ce nom ne doit pas comprendre plus de 255 caractères. Il faut éviter les noms réservés du langage (*Dim*, *Integer*,...) et **il est interdit de placer des espaces dans le nom des variables.**

La portée d'une variable est limitée par la zone où elle est définie. Les types de variables en VBA sont les suivants : **Boolean, Currency, Date, Byte, Integer, Long, Single, Double, Object, String et Variant.**

Il faut cependant savoir que le VBA ne nous oblige pas à définir les variables que nous utilisons sauf si on le demande explicitement. Nous vous avons demandé de forcer VBA à définir toutes les variables, ce qui produit l'instruction ***Option explicit*** au début de la fenêtre de code.

Mais si on ne force pas VBA à définir les variables, il attribue alors le type ***Variant*** à la variable. Ce type permet de contenir n'importe quel autre type de variables. Ce faisant, il faut cependant savoir que la mémoire utilisée sera alors en général plus grande que si on précise le type de variable désirée.

On déclare les variables au moyen de l'instruction ***Dim*** : Par exemple,

```
Dim maVari As Integer
```

```
Dim uneAutreVariable As String, maPlage As Range("E1:E4")
```

Mais

```
Dim i,j,k As Integer
```

Déclare i comme un *Integer* mais j et k seront de type *Variant*....

Il est aussi possible de déclarer des variables de type *String* de longueur fixe comme dans l'exemple suivant :

```
Dim unNumTel As String*10
```

Pour initialiser des variables numériques, on n'a qu'à déposer une valeur numérique dans celle-ci comme l'illustre l'exemple suivant :

```
maVari = 2;
```

Pour initialiser une variable de type *String* vous placez le texte entre deux guillemets :

```
uneAutreVariable = "Allo"
```

Enfin, lorsque l'on veut initialiser des variables de type *objet* on utilise la syntaxe suivante :

```
Set maPlage = Range("D4:D19")
```

Voici maintenant comment on réinitialise différents types de variables :

```
maVari=0
```

```
uneAutreVariable = ""
```

```
Set maPlage = Nothing
```

Une variable qui contient des dates peut contenir des dates comprises entre le 1 janvier 100 et le 31 décembre 9999 (!). Et on entre les dates de la façon suivante :

```
Dim UneDate As date = #12/1/2013#
```

**Prenez note que les dates que l'on entre en VBA doivent être du format US (mm/jj/année) mais lorsqu'on les fait afficher dans une boîte de message, elles se présenteront sous le format régional de l'ordinateur.**

Voici une procédure VBA qui démontre cela :

```
Sub AfficheDate()
```

```
Dim UneDate As Date
```

```
UneDate = #2/23/2013#
```

```
MsgBox ("La date est le " & UneDate)
```

```
End Sub
```

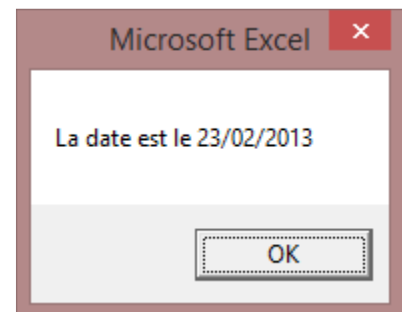


Figure 20

Pour exécuter une telle procédure, on place d'abord le curseur à l'intérieur de la procédure et on choisit le menu **exécuter** ou on appuie sur la touche **F5**. On verra s'afficher la boîte de dialogue ci-dessus. On remarque une différence entre la valeur de la date dans le code et celle affichée.

Lorsqu'une variable est définie dans une procédure, sa portée est alors limitée à la procédure. Sa valeur n'est accessible que dans la procédure et sera remise à zéro entre chaque appel de la procédure. Pour qu'une variable conserve sa valeur entre les appels de la procédure, il faut la définir avec l'attribut **static**.

Si on veut que la variable soit connue de toutes les procédures, il faut alors la déclarer dans la section **Générale** du code comme l'illustre la figure suivante :

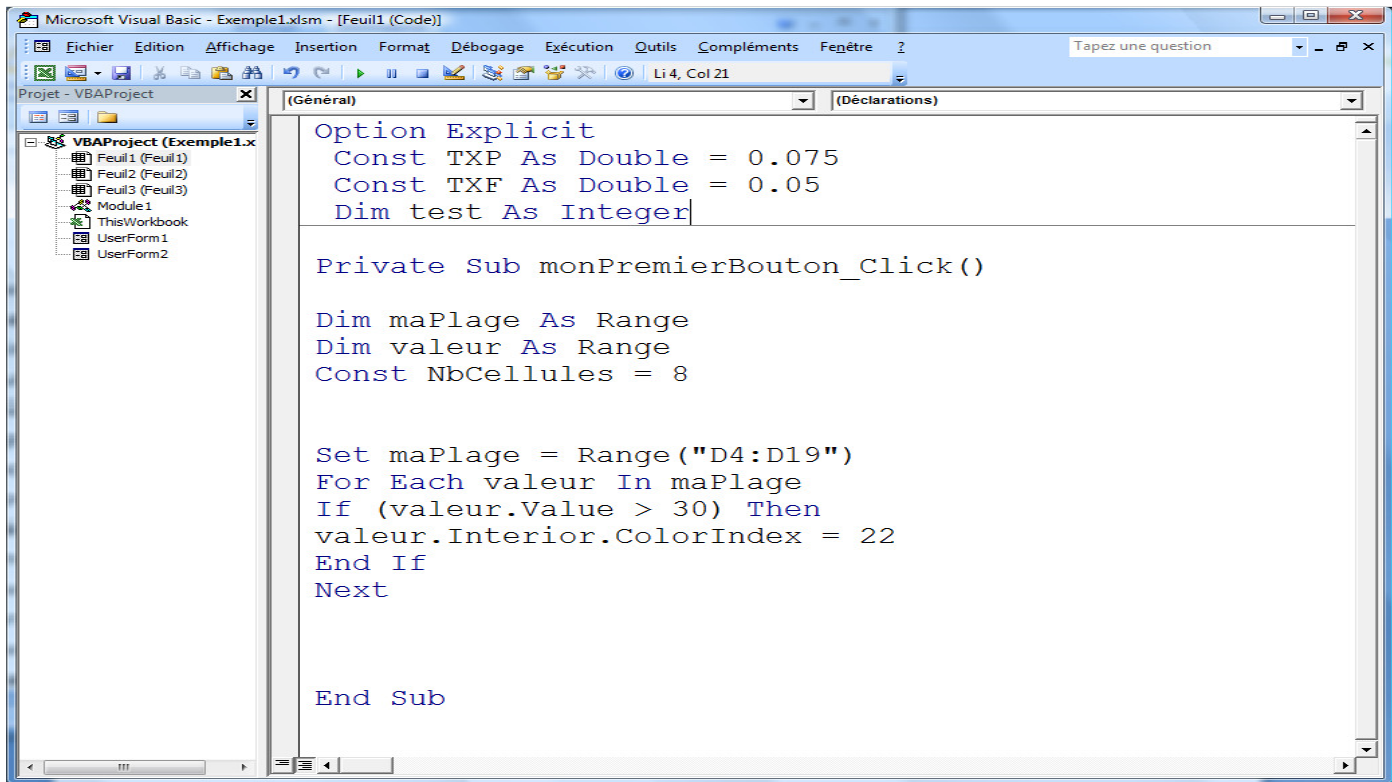


Figure 21

Dans la figure précédente, on retrouve deux constantes définies dans la zone générale du module ainsi qu'une définition de variable, la variable *test*. Ces constantes et cette variable seront visibles de toutes les procédures de ce module alors que les variables déclarées dans la procédure *monPremierBouton\_Click()* ne seront accessibles qu'à l'intérieur de cette procédure.

## IV. Les structures de décision et de répétition

### i. Le If

En VBA, on rencontre la structure *if* qui possède la syntaxe suivante :

```
If condition Then
    Instructions..
[Else
    Instructions]
End if
```

Dans le paragraphe précédent, le texte se trouvant entre crochets est facultatif. Voici un exemple de If:

```
If prix > 100 Then prix=prix-prix*0.05
    test = Range("B4").Value
    If test >= 10 Then
        Largeur = 1000*Valeur
        Longueur = 20*Valeur
    Else
        Largeur = 100 * Valeur
        Longueur = 10 * Valeur
    End if
```

## ii. Le Select Case

On rencontre aussi la structure Select Case dont la syntaxe est la suivante :

### *Select Case Condition*

#### *Case expression 1*

*Instructions à exécuter*

#### *Case expression 2*

*Instructions à exécuter*

...

#### *Case Else*

*Instructions à exécuter si aucune n'est vraie*

### *End Select*

Voici un exemple d'utilisation de cette instruction :

```
Private Sub cmdComission_Click()
    Dim Vente As Single
    Dim TauxCom As Single
    Vente = Feuil1.Range("H10")

    Select Case Vente
        Case Is <= 1000
            TauxCom = 0.01
        Case Is <= 2000
            TauxCom = 0.02
        Case Is <= 3000
            TauxCom = 0.03
        Case Is <= 4000
            TauxCom = 0.04
        Case Is <= 5000
            TauxCom = 0.05
        Case Else
            TauxCom = 0.0
    End Select
    Feuil1.Range("H11").Value = TauxCom
End Sub
```

### iii. La structure de répétition For Next

Cette structure de répétition permet de répéter plusieurs fois certaines parties du code. On l'utilise lorsque l'on connaît à l'avance le nombre de répétitions à effectuer.

**For compteur = début To fin [Step incrément]**

*Instructions...*

**Next compteur**

Dans cette structure, on donne une valeur de début (*début*) à la variable *compteur* et une valeur de fin (*fin*). La variable *compteur* est la variable de contrôle de la boucle. Si on n'utilise pas l'attribut *Step* qui permet de préciser la valeur de l'incrément (ou le pas), celui-ci sera de 1. Voici un exemple :

```
For i = 1 to 20
    total = total * 2
Next i
```

### iv. La structure de répétition For Each

La structure de répétition *For Each* permet de parcourir tous les éléments d'un tableau ou d'une collection. Sa syntaxe est la suivante :

**For Each éléments dans la collection**

*Instructions...*

**Next éléments**

On peut ainsi parcourir toute une collection (ou un tableau) sans avoir à préciser son point de départ ni sa longueur. Voir le deuxième exemple de la page 12.

### v. La boucle Do While et Do Until

Voici une autre forme de structure de répétition qui exécute des instructions tant qu'une condition n'est pas remplie. Voici la syntaxe de ces instructions :

**Do**

*Instructions*

**Loop While Condition**                      et

**Do Until Condition**

*Instructions*

**Loop**

## 5. Les formulaires

Nous allons maintenant présenter le formulaire qui permet de créer une boîte de dialogue personnalisée. Nous avons vu précédemment qu'il était possible de placer des boutons sur une feuille de calcul et d'effectuer des actions. Il est souvent préférable de ne pas afficher ces boutons en permanence mais plutôt de les regrouper dans des formulaires distincts que l'on peut appeler au besoin. Un formulaire est en quelque sorte une boîte de dialogue permettant d'effectuer des tâches diverses. Nous allons les utiliser avec des feuilles *Excel*. Nous allons les introduire au moyen d'un exemple. Soit la feuille de calcul suivante :

	A	B	C	D	E	F	G	H
1								
2			Sami	Lucie	Ian	Bertin	Total Mensuel	
3		MOIS						
4		janvier	12345,34	234,45	2345,23	1234,56	16159,58	
5		février	653,3	345,67	239,12	982,34	2220,43	
6		mars	319,4	3902,45	23,24	2100,34	6345,43	
7		avril	190,34	234,72	267,34	2900,34	3592,74	
8		mai	34,56	12345,67	789,56	892,13	14061,92	
9		juin	34,56	234,56	209,34	23,45	501,91	
10		juillet	234	2456,89	244,12	2309,89	5244,9	
11		août	340,56	234,56	34,67	9087,34	9697,13	
12		septembre	239,9	234,12	12987,34	23,23	13484,59	
13		octobre	34,45	234,56	23,34	34,67	327,02	
14		novembre	127,34	390,12	21,34	199,34	738,14	
15		décembre	120,56	234,56	234,45	1239823	1240412,57	
16								
17		Total associé	14674,31	21082,33	17419,09	1259610,63		
18								
19					Grand total	1312786,36		

Figure 22

Nous allons ajouter un formulaire à cette feuille de calcul afin de faire afficher le total des ventes d'un associé sur un certain nombre de mois consécutifs. On commence par se placer en mode développeur et on clique sur l'icône *Visual Basic* :



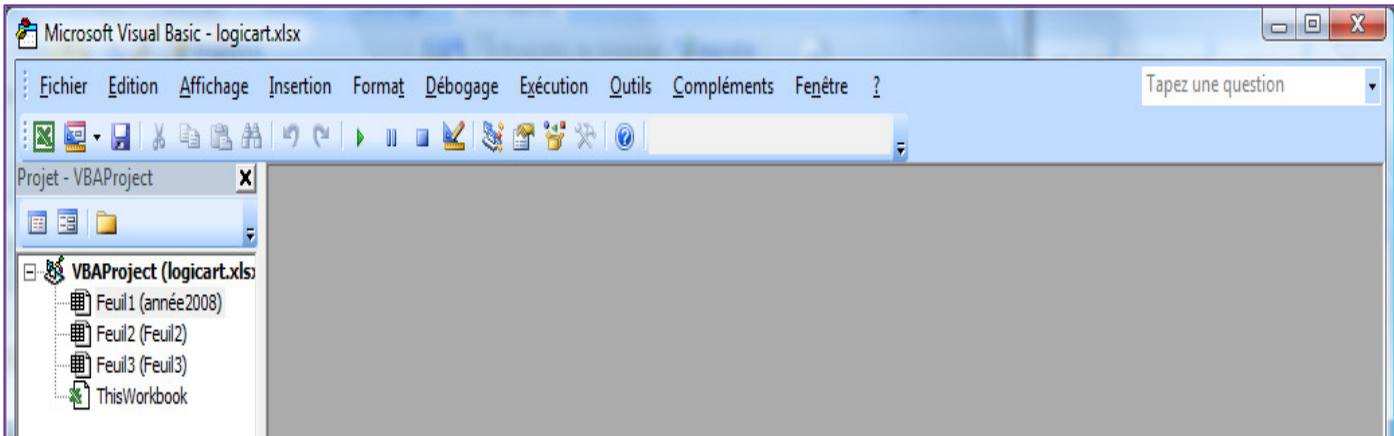


Figure 23

On remarque alors l'apparition de la fenêtre contenant les éléments du **VBAProject**. Si elle ne se présente pas, appuyez sur **Ctrl+R**. On y voit ici entre autres les trois feuilles du classeur qui se présentent dans les versions antérieures d'*Excel*. On clique alors sur le menu *Insertion* et on choisit d'insérer un *UserForm* qui est en fait le formulaire. On verra alors la figure suivante :

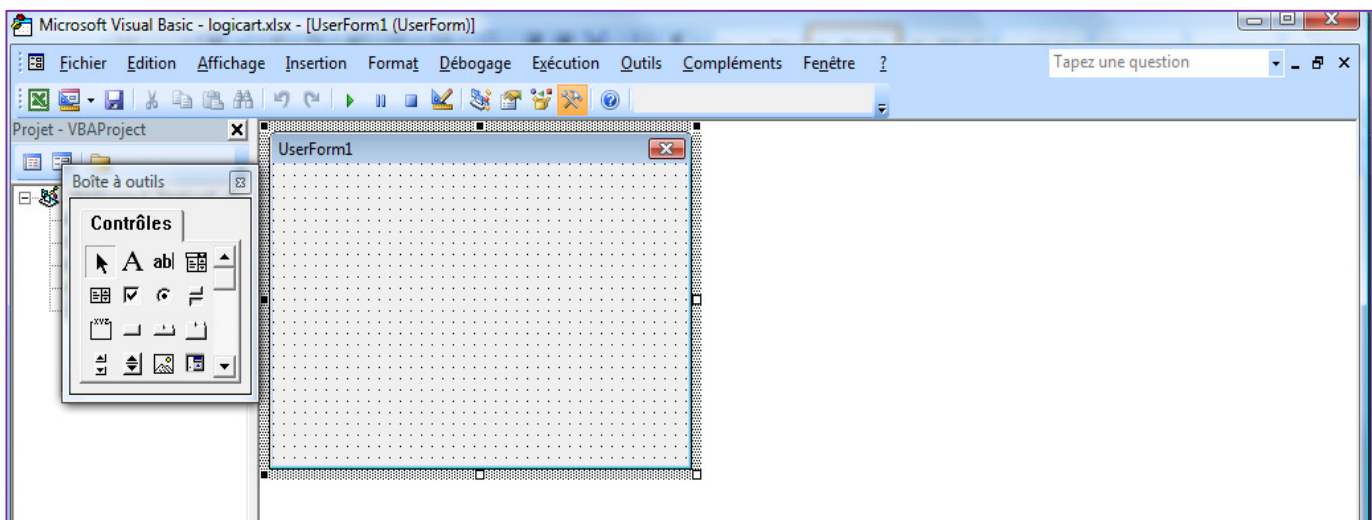


Figure 24

On y remarque une feuille nommée **UserForm1** ainsi qu'une palette de contrôles similaires à ceux déjà rencontrés dans notre apprentissage du VBA.

Après avoir sélectionné la boîte **UserForm1**, on change sa propriété **caption** afin d'y placer le titre TotalVentes. On y insère deux contrôles **label**, un contrôle **listbox** et un bouton en disposant le tout comme le montre la figure suivante :

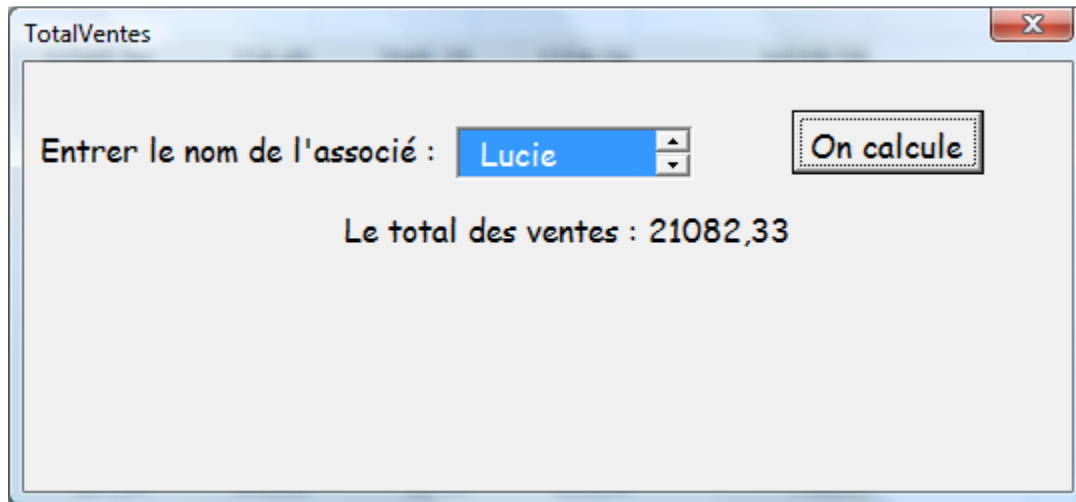


Figure 25

Le code permettant d'afficher les ventes d'un associé après l'avoir choisi d'une liste déroulante est le suivant :

#### Option Explicit

```
Public Sub cmdEvalue_Click()  
    lblRep.Caption = Application.WorksheetFunction.Sum(Range(lstAssociés.Value))  
End Sub  
  
Private Sub UserForm_Activate()  
    Me.lstAssociés.AddItem ["Sami"]  
    Me.lstAssociés.AddItem ["Lucie"]  
    Me.lstAssociés.AddItem ["Ian"]  
    Me.lstAssociés.AddItem ["Bertin"]  
End Sub
```

Ce formulaire fait appel à un contrôle *listBox* qui est rempli lorsque le formulaire est activé. Nous en discuterons en classe.

Les formulaires peuvent s'afficher en mode **modal** ou **non modal**. En mode modal, l'exécution du formulaire devra se terminer avant d'être en mesure d'exécuter d'autres lignes de code ou encore d'utiliser des fonctionnalités d'Excel. En mode non modal, les formulaires ne bloquent rien. On démontre cela au moyen de l'ajout de deux boutons sur la feuille Excel précédente. Un clic sur le premier bouton produira un affichage du formulaire en mode modal (*formulaire.Show*), ce qui bloquera l'affichage de la boîte de message qui suit tant que le formulaire ne sera pas fermé. Le deuxième bouton, *Affichage non modal du formulaire* affichera le formulaire (*formulaire.Show False*) mais la boîte de message s'affichera elle aussi. On utilise ces fonctionnalités au besoin :

**Figure 26**

### Option Explicit

Lorsque l'on veut afficher un formulaire, on fait donc appel à la méthode **Show**. La méthode **Hide** permet de cacher un formulaire tout en préservant son espace mémoire. On utilise cette instruction si on veut conserver l'état actuel du formulaire pour utilisation ultérieure, par exemple pour conserver les valeurs qu'il contient déjà. Si vous voulez plutôt rappeler le formulaire à neuf, utilisez plutôt la méthode **Load**. Et à la différence de la méthode **Hide** la méthode **Unload** libère l'espace mémoire occupée par le formulaire. On peut deviner de l'exemple précédent qu'il est possible de construire de multiples formulaires qui pourront être appelé au besoin au moyen de boutons sur la feuille de calcul.

## 6. Exemples de codes

### I. Problèmes

#### i. Exemple Base de données- Liste des enseignants de l'institut Teccart

Ouvrir le classeur contenant la feuille Base de donnée 1 et nommer une nouvelle feuille Final. Insérer un bouton dans la feuille Base de donnée1. Ecrire une procédure qui, en cliquant sur ce bouton, permet de faire automatiquement les actions suivantes dans la feuille Final:

1. Afficher les colonnes C, D, E, I de la feuille Base de donnée 1 dans l'ordre D, C, I, E dans la feuille Final :

D -----> A

C -----> B

I -----> C

E -----> D

2. Mettre l'entête des colonnes de la feuille Final, en gras et centrés.

3. Attribuer une couleur différente aux lignes de la feuille Final, selon l'ancienneté des enseignants :

Ancienneté	Couleur	Code
0 à 5	Vert clair	5296274
6 à 10	Jaune	65535
11 à 20	Bleu clair	15773696
21 à 30	Rouge	255
31 et +	Jaune clair	13434879

4. Afficher la moyenne de l'ancienneté à la fin de la colonne C.

#### ii. Exemple Population du Canada

Ouvrir le classeur contenant la feuille Canada-Population et écrire une procédure affichant:

1. les provinces du Canada avec une structure For Next. Refaire la procédure avec une structure Do Loop.
2. les abréviations des provinces et des territoires du Canada.



## II. Les codes

### i. Exemple Base de données

Sub Bouton ()

' INTERVERTIR LES COLONES

Sheets("Base de donnée 1").Select 'selectionner la feuille Base de données 1  
Range("D4").Select 'selectionner la cellule D4  
Range(Selection, Selection.End(xlDown)).Select 'selectionner la colonne D du début à la fin  
Selection.Copy ' Copier la colonne D

Sheets("Final").Select 'Aller à la feuille Final  
Range("A1").Select 'selectionner la cellule A1  
ActiveSheet.Paste ' coller la colonne D dans la colonne A

Sheets("Base de donnée 1").Select 'selectionner la feuille Base de données 1  
Range("C4").Select 'selectionner la cellule C4  
Range(Selection, Selection.End(xlDown)).Select 'selectionner la colonne C du début à la fin  
Selection.Copy ' Copier la colonne C

Sheets("Final").Select 'Aller à la feuille Final  
Range("B1").Select 'selectionner la cellule B1  
ActiveSheet.Paste ' coller la colonne C dans la colonne B

Sheets("Base de donnée 1").Select 'selectionner la feuille Base de données 1  
Range("I4").Select 'selectionner la cellule I4  
Range(Selection, Selection.End(xlDown)).Select 'selectionner la colonne I du début à la fin  
Selection.Copy ' Copier la colonne I

Sheets("Final").Select 'Aller à la feuille Final  
Range("C1").Select 'selectionner la cellule C1  
ActiveSheet.Paste ' coller la colonne I dans la colonne C

Sheets("Base de donnée 1").Select 'selectionner la feuille Base de données 1  
Range("E4").Select 'selectionner la cellule D4  
Range(Selection, Selection.End(xlDown)).Select 'selectionner la colonne E du début à la fin  
Selection.Copy ' Copier la colonne E

Sheets("Final").Select 'Aller à la feuille Final  
Range("D1").Select 'selectionner la cellule A1  
ActiveSheet.Paste ' coller la colonne E dans la colonne D

### ' ENTETE EN GRAS ET CENTRÉ

```
Sheets("Final").Select  
Range("A1", "D1").Select  
With Selection  
    .HorizontalAlignment = xlCenter  
    .VerticalAlignment = xlCenter
```

```
End With
```

```
Selection.Font.Bold = True
```

### ' COLORER LES LIGNES

```
Sheets("Final").Select
```

```
FinLigne = ActiveSheet.UsedRange.Rows.Count + 1 ' Compter le nombre de lignes dans la colonne jusqu'à la fin du tableau
```

```
NumeroLigne = 2 ' Première ligne du tableau ( de la colonne)
```

```
While NumeroLigne < FinLigne
```

```
    Cellule = Range("C" & NumeroLigne).Value
```

```
    Select Case Cellule
```

```
        Case Is <= 5
```

```
            Range("A" & NumeroLigne, "D" & NumeroLigne).Interior.Color = 5296274 ' attribuer la couleur vert clair
```

```
        Case Is <= 10
```

```
            Range("A" & NumeroLigne, "D" & NumeroLigne).Interior.Color = 65535 ' attribuer la couleur jaune
```

```
        Case Is <= 20
```

```
            Range("A" & NumeroLigne, "D" & NumeroLigne).Interior.Color = 15773696 ' attribuer la couleur bleu clair
```

```
        Case Is <= 30
```

```
            Range("A" & NumeroLigne, "D" & NumeroLigne).Interior.Color = 255 ' attribuer la couleur rouge
```

```
        Case Else
```

```
            Range("A" & NumeroLigne, "D" & NumeroLigne).Interior.Color = 13434879 ' attribuer la couleur Jaune clair
```

```
    End Select
```

```
    NumeroLigne = NumeroLigne + 1
```

```
Wend
```

' FAIRE LA MOYENNE DE LA COLONNE C

Range("C2").Select

FinLigneC = ActiveSheet.UsedRange.Rows.Count + 1 ' Compter le nombre de lignes dans la colonne jusqu'à la fin du tableau

Range("C" & FinLigneC).Value = Application.WorksheetFunction.Average(Range("C2", "C" & FinLigneC)) ' Formule de la Moyenne

Range("C" & FinLigneC).Font.Bold = True ' la réponse en gras

' Range("C" & FinLigneC).Style = "Currency" ' la réponse en monétaire

End Sub

## ii. Exemple Population du Canada

Sub ExempleForNext()

Dim NuméroLigne As Integer

Dim Provinces As String

For NuméroLigne = 5 To 14

    If Cells(NuméroLigne, 3).Value = "" Then

        Exit For

    Else

        Provinces = Provinces & Cells(NuméroLigne, 3).Value & vbCrLf ' ajoute ligne vide à la fin

    End If

Next NuméroLigne

MsgBox Provinces

End Sub

Sub ExempleDoLoop()

Dim NuméroLigne As Integer

Dim Provinces As String

NuméroLigne = 5

Do While Cells(NuméroLigne, 3).Value <> "Territoire du Nord Ouest"

    Provinces = Provinces & Cells(NuméroLigne, 3).Value & vbCrLf ' vbCrLf force une ligne à la fin



```

        NuméroLigne = NuméroLigne + 1
    Loop
    MsgBox Provinces

End Sub

Sub ExempleForEachNext()
    Dim Cellule As Range
    Dim ProvincesEtTerritoires As String
    For Each Cellule In Range("D5:D17").Cells
        If Cellule.Value = "" Then
            Exit For
        Else
            ProvincesEtTerritoires = ProvincesEtTerritoires & Cellule.Value & vbCrLf
        End If
    Next Cellule
    MsgBox ProvincesEtTerritoires

End Sub

```