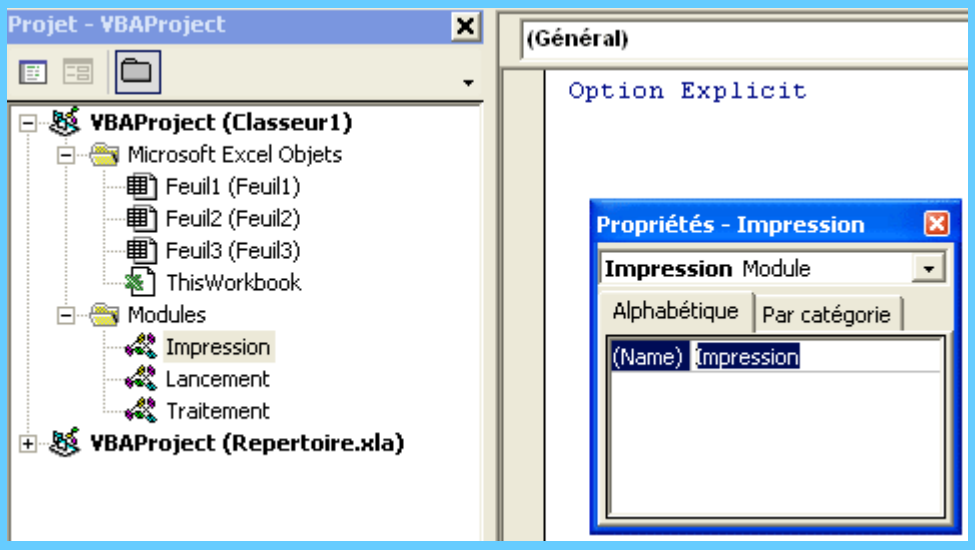
Cours VBA - Présentation -

Le code VBA s'écrit dans les modules à l'intérieur de procédures ou de fonctions.
Dans VBE, créez un nouveau module par le menu "Insertion - Module". Renomez le module à l'aide de la fenêtre propriétés, la recherche de vos procédures sera plus rapide.



Une procédure est une suite d'instructions effectuant des actions. Elle commence par Sub + NomDeLaProcédure et se termine par End Sub. Le nom des procédures ne doit pas commencer par une lettre et ne doit pas contenir d'espaces. Utilisez le caractère de soulignement pour séparer les mots. Je vous conseille de les écrire comme des noms propres.
Pour déclarer une procédure, taper Sub et son nom puis taper Entrée. VBE ajoute automatiquement les parenthèses et la ligne End Sub.
Exemple de Procédure nommée Essai :

```
Sub Essai()
    MsgBox "Bonjour"
End Sub
```

Une fonction est une procédure qui renvoie une valeur. Elle se déclare de la même façon qu'une procédure.
Exemple de fonction nommée Calcul :

```
Function Calcul(Nbre1 As Integer, Nbre2 As Integer)
    Calcul = Nbre1 + Nbre2
End Function
```

En général, on écrit une instruction par ligne.
Il est possible d'ajouter des lignes de commentaire entre les lignes d'instruction ou au bout de celles-ci. Les commentaires sont précédés d'une apostrophe et prennent une couleur différente (définie dans les options de VBE) :

```
Sub Essai()
    Dim Invite as String 'Nom de l'utilisateur
    Invite = "Toto"
    'Message bonjour à l'utilisateur
    MsgBox "Bonjour " & Invite
End Sub
```

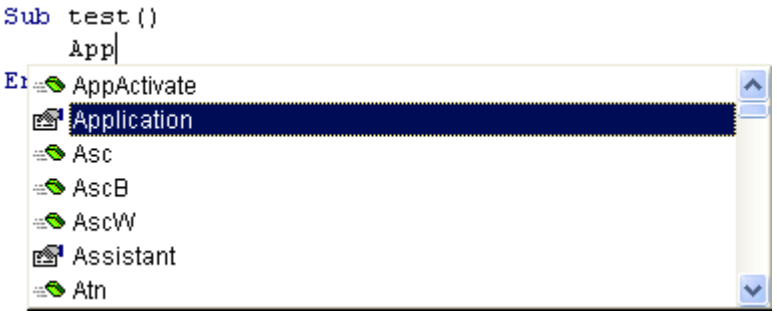
Résultat :

Il n'y a pas de limite de caractères pour chaque ligne d'instruction. Il est toutefois possible d'écrire une instruction sur plusieurs lignes afin d'augmenter la visibilité du code. Pour cela, il faut ajouter le caractère de soulignement avant le passage à la ligne (touche Entrée) :

```
Sub Essai()
    MsgBox("Aujourd'hui nous sommes le " _
    & Date, vbInformation, "Mon Application")
End Sub
```

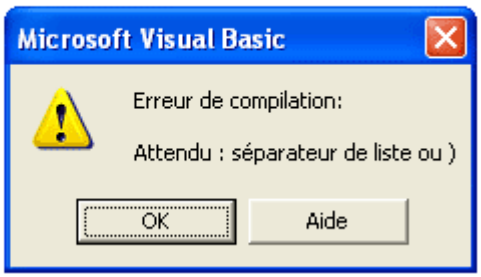
Résultat :

L'option "Info express automatique" permet d'afficher les informations de la fonction que vous venez de taper. Il est également possible d'obtenir de l'aide à tout moment par la combinaison de touches Crtl+j :



La vérification automatique de la syntaxe vous alerte si il y a une erreur dans l'écriture du code et la ligne de code change de couleur . Si la vérification automatique de la syntaxe n'est pas activée, la boite d'alerte ne s'affiche pas.

```
Sub Test()
    range("A1) = 1

End Sub
```



Chaque procédure Sub ou Function peut être appelée de n'importe qu'elle autre procédure du projet. Pour restreindre la portée d'une procédure au module, déclarez-la en private :

```
Private Sub Essai()
    MsgBox "Bonjour"
End Sub

Private Function Calcul(Nbre1, Nbre2)
    Calcul = Nbre1 + Nbre2
End Function
```

A l'intérieur de vos procédures, écrivez vos instructions en minuscules, VBE se chargera de transformer votre code par des majuscules.

Il existe souvent de multiples façons d'arriver à un résultat.Une bonne analyse des tâches à accomplir est nécessaire avant de se lancer dans la création d'une application.
Si vous n'avez aucune expérience en VBA, vous verrez que l'on y prend vite goût et que l'on arrive très rapidement à de surpenants résultats.

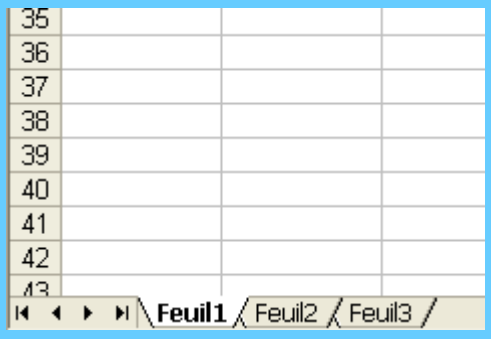
HAUT

FERMER

Cours VBA - Le vocabulaire -

VBA manipule les objets de l'application hôte. Chaque objet possède des propriétés et des méthodes.

Les objets :
Chaque objet représente un élément de l'application. Sous Excel, un classeur, une feuille de calcul, une cellule, un bouton, etc ... sont des objets. Par exemple, Excel représente l'objet Application, Workbook l'objet classeur, Worksheet l'objet feuille de calcul etc...
Tous les objets de même type forment une collection comme, par exemple, toutes les feuilles de calcul d'un classeur. Chaque élément est alors identifié par son nom ou par un index.



Pour faire référence à la Feuil2, on va utiliser Worksheets(2) ou Worksheets("Feuil2")

Chaque objet peut avoir ses propres objets. Par exemple, Excel possède des classeurs qui possèdent des feuilles qui possèdent des cellules. Pour faire référence à une cellule, on pourrait ainsi utiliser :

```
Application.Workbooks(1).Worksheets("Feuil2").Range("A1")
```

Les propriétés :
Une propriété correspond à une particularité de l'objet. La valeur d'une cellule, sa couleur, sa taille, etc...sont des propriétés de l'objet Range. Les objets sont séparés de leurs propriétés par un point. On écrira ainsi Cellule.Propriété=valeur :

```
'Mettre la valeur 10 dans la cellule A1
Range("A1").Value = 10
```

Une propriété peut également faire référence à un état de l'objet. Par exemple, si on veut masquer la feuille de calcul "Feuil2", on écrira :

```
Worksheets("Feuil2").Visible = False
```

Les méthodes :
On peut considérer qu'une méthode est une opération que réalise un objet. Les méthodes peuvent être considérées comme des verbes tels que ouvrir, fermer, sélectionner, enregistrer, imprimer, effacer, etc... Les objets sont séparés de leurs méthodes par un point.

Par exemple, pour sélectionner la feuille de calcul nommé "Feuil2", on écrira :

```
Worksheets("Feuil2").Select
```

Lorsque l'on fait appel à plusieurs propriétés ou méthodes d'un même objet, on fera appel au bloc d'instruction **With** Objet *Instructions* **End With.** Cette instruction rend le code souvent plus facile à lire et plus rapide a exécuter.

```
'Mettre la valeur 10 dans la cellule A1, la police en gras
et en italique et copier la cellule.
With Worksheets("Feuil2").Range("A1")
    .Value = 10
    .Font.Bold = True
    .Font.Italic = True
    .Copy
End With
```

Ce vocabulaire peut paraître déroutant mais deviendra très rapidement familier lors de la création de vos premières applications.

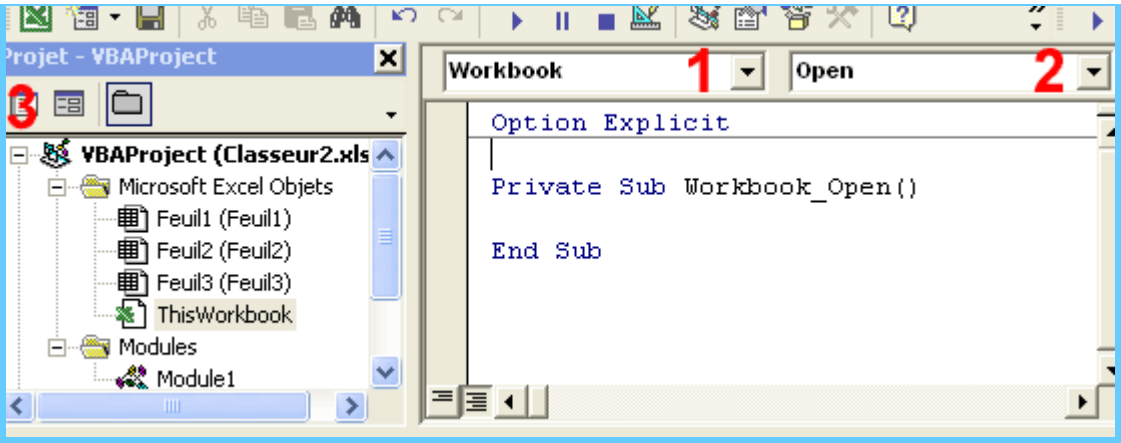
Cours VBA - Les évènements -

Pour qu'une macro se déclenche, il faut qu'un évènement (un clic sur un bouton, l'ouverture d'un classeur, etc...) se produise. Sans évènements, rien ne peut se produire.

Les évènements liés aux objets.
Les principaux objets pouvant déclencher une macro sont :

- Un classeur
- Une feuille de travail
- Une boite de dialogue

Chacun de ces objets possède leur propre module. Pour y accéder, lancer l'éditeur de macro :



Pour créer une procédure évènementielle liée à un classeur, sélectionner le classeur "ThisWorkbook" puis cliquez sur l'icône **3** (ou plus simplement double-clic sur "ThisWorkbook").
Vous accédez ainsi au module lié à l'objet. Sélectionnez "Workbook" dans la liste **1** puis sur l'évènement désiré dans la liste **2**.
Par exemple, le code suivant lancera la procédure nommée "Test" à l'ouverture du classeur :

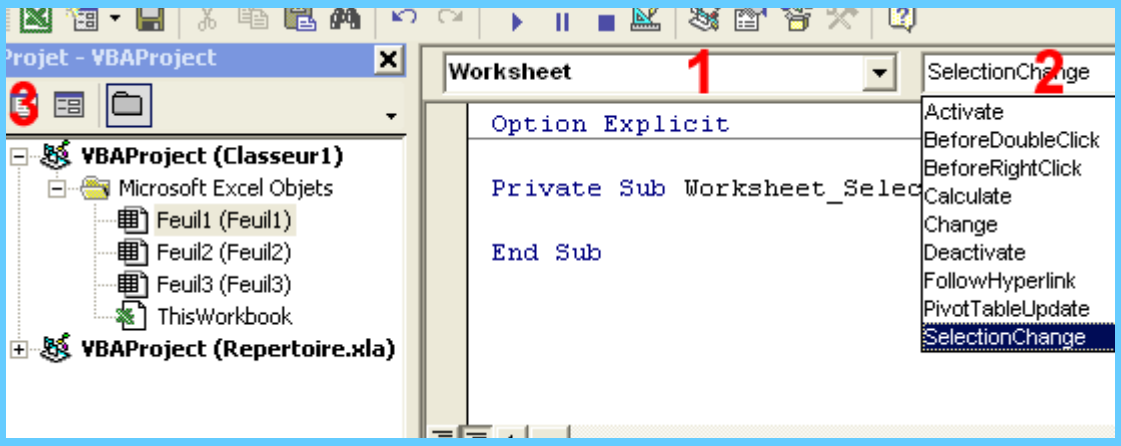
```
Private Sub Workbook_Open()
    Test
End Sub
```

Liste des évènements de l'objet Workbook :.

Evénements:	*Se produit :*
Activate	quand le classeur ou une feuille est activé
AddinInstall	quand le classeur est installé en macro complémentaire
AddinUninstall	quand le classeur est désinstallé en macro complémentaire
BeforeClose	avant que le classeur soit fermé
BeforePrint	avant l'impression du classeur
BeforeSave	avant l'enregistrement du classeur
Deactivate	quand le classeur ou une feuille est désactivé
NewSheet	lorsqu'une nouvelle feuille est créée
Open	à l'ouverture du classeur
PivotTableCloseConnection	lorsqu'un qu'un rapport de tableau croisé dynamique se déconnecte de sa source de données
PivotTableOpenConnection	lorsqu'un qu'un rapport de tableau croisé dynamique se connecte à une source de données
SheetActivate	lorsqu'une feuille est activée
SheetBeforeDoubleClick	lors d'un double-clic
SheetBeforeRightClick	lors d'un clic avec le bouton droit de la

	souris
SheetCalculate	après le recalcul d'une feuille de calcul
SheetChange	lors de la modification d'une cellule
SheetDeactivate	lorsqu'une feuille est désactivée
SheetFollowHyperlink	lors d'un clic sur un lien hypertexte
SheetPivotTableUpdate	lors de la mise à jour de la feuille du rapport de tableau croisé dynamique
SheetSelectionChange	lors d'un changement de sélection sur une feuille de calcul
WindowActivate	lorsqu'un classeur est activé
WindowDeactivate	lorsqu'un classeur est désactivé
WindowResize	lors du redimentionnement de la fenêtre d'un classeur

La création d'une procédure évènementielle liée à une feuille de calcul se fait de la même façon.



Liste des évènements de l'objet Worksheet :

Evénements:	*Se produit :*
Activate	quand une feuille est activée
BeforeDoubleClick	lors d'un double-clic
BeforeRightClick	lors d'un clic avec le bouton droit de la souris
Calculate	après le recalcul de la feuille de calcul
Change	lors de la modification d'une cellule
Deactivate	quand une feuille est désactivée
FollowHyperlink	lors d'un clic sur un lien hypertexte
PivotTableUpdate	lorsqu'un rapport de tableau croisé dynamique a été mis à jour
SelectionChange	lors d'un changement de sélection

Certaines procédures évènementielles possèdent des paramètres tels que "**Cancel**", qui peut

annuler la procédure, "**SaveAsUi**" qui, dans la procédure "Workbook_BeforeSave" affiche la boite "Enregistrer sous", "**Sh**" qui représente la feuille de calcul, "**Target**" qui représente l'objet sélectionné(Cellule, graphique, lien hypertexte), "**Wn**" qui représente la fenêtre active.

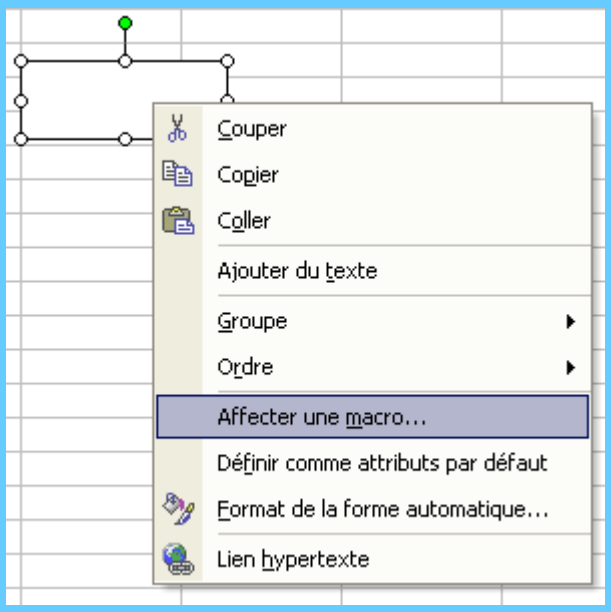
Par exemple, le paramètre "Cancel", peut annuler la procédure. Pour empêcher l'impression du classeur, on utilisera :

```
Private Sub Workbook_BeforePrint(Cancel As Boolean)
    Cancel = True
End Sub
```

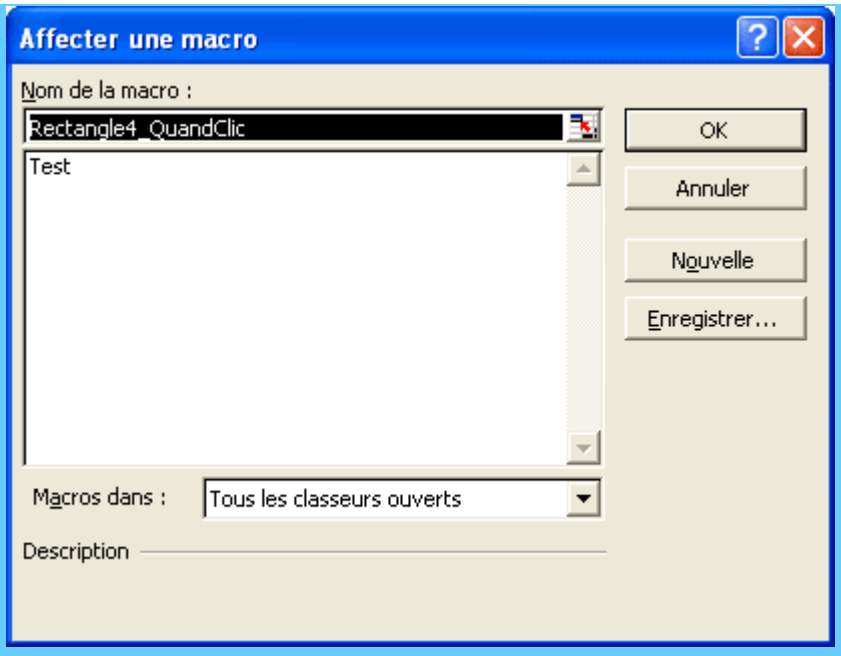
Pour récupérer la valeur d'une cellule modifiée, on utilisera :

```
Private Sub Worksheet_Change(ByVal Target As Range)
    MsgBox Target.Value
End Sub
```

Une macro peut également se déclencher en cliquant sur un élément graphique de l'application (Une image, une zone de texte, un objet WordArt, un rectangle ...). Créez un élément puis cliquez sur "Affecter une macro" dans le menu contextuel.



Cliquez sur le nom de la macro désirée puis validez.



Un simple clic sur l'objet lancera la macro.

Il existe également des procédures évènementielles liées aux boites de dialogues (Voir le cours sur les UserForms).

Les évènements non liés aux objets.

Une macro peut également être déclenchée à une heure donnée (OnTime) ou lorsque l'utilisateur appuie sur une touche (OnKey).

Le déclenchement d'une macro nommée "Test" à 15 Heures se fait par la ligne d'instruction suivante :

```
Application.OnTime TimeValue("15:00:00"), "Test"
```

Le déclenchement d'une macro nommée "Test" lorsque l'utilisateur appuie sur la touche "F1" se fait par la ligne d'instruction suivante :

```
Application.OnKey "{F1}", "Test"
```

Liste des codes correspondant aux touches:

Touches:	*Codes :*
AIDE	{HELP}
ATTN	{BREAK}
BAS	{DOWN}
DÉBUT	{HOME}
DÉFILEMENT	{SCROLLLOCK}

```
                    DROITE {RIGHT}
                     ºCHAP {ESCAPE} ou {ESC}
                  EFFACER {CLEAR}
ENTRºE(pavé numérique) {ENTER}
                    ENTRºE ~
                F1 à F15 {F1} à {F15}
                     FIN {END}
                   GAUCHE {LEFT}
                     HAUT {UP}
                INSERTION {INSERT}
          PAGE PRºCºDENTE {PGUP}
           PAGE SUIVANTE {PGDN}
                 RET.ARR {BACKSPACE} ou {BS}
                  RETOUR {RETURN}
  SUPPRESSION ou SUPPR {DELETE} ou {DEL}
             TABULATION {TAB}
               VERR.MAJ {CAPSLOCK}
               VERR.NUM {NUMLOCK}
```

Il est possible de combiner les touches avec "Alt" en insérant le caractère "%" ou avec "Ctrl" en insérant le caractère "^" ou avec la touche "MAJ" en insérant le caractère "+". Ainsi le déclenchement d'une macro nommée "Test" lorsque l'utilisateur appuie sur la combinaison de touches "Ctrl+MAJ+F1" se fait par la ligne d'instruction suivante

```
Application.OnKey "^+{F1}", "Test"
```


HAUT

FERMER

Cours VBA - Les messages -

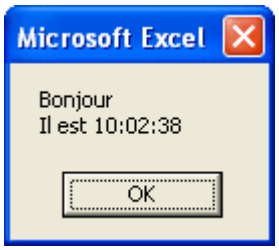
Lors d'une procédure, les messages servent à communiquer avec l'utilisateur.
Il existe des messages qui donnent de l'information et d'autres qui en demandent.

Les MsgBox
Les MsgBox peuvent simplement donner une information. La procédure est alors stoppée tant que l'utilisateur n'a pas cliqué sur le bouton.

```
MsgBox "Bonjour"
```

Le texte peut-être affiché sur plusieurs lignes en utilisant le code retour chariot chr(13) ou le code retour ligne chr(10).

```
MsgBox "Bonjour" & Chr(10) & "Il est " & Time
```

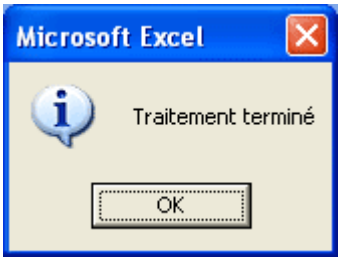


Vous pouvez ajouter une icône concernant le type de message à afficher.
Les types d'attribut icône :

Constante :	*Icône :*	
vbCritical		Pour une erreur fatale
vbExclamation		Pour une remarque
vbInformation		Pour une information
vbQuestion		Pour une question

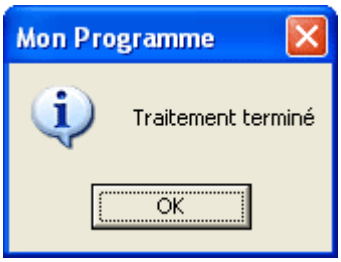
La syntaxe pour ajouter une icône est MsgBox "Message", attribut icône :

```
MsgBox "Traitement terminé", vbInformation
```



Le titre de la fenêtre (Microsoft Excel) peut être changé. La syntaxe est : MsgBox "Message", attribut icône, "Titre de la fenêtre" :

```
MsgBox "Traitement terminé", vbInformation, "Mon Programme"
```

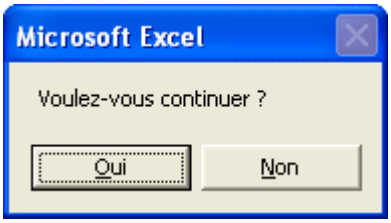


Les MsgBox peuvent également demander une information à l'utilisateur. Dans ce cas, la boite de message comprend plusieurs boutons
Les types d'attribut Boutons :

Constante :	*Boutons* :		
vbAbortRetryIgnore	Abandonner	Recommencer	Ignorer
vbOKCancel	OK	Annuler	
vbRetryCancel	Recommencer	Annuler	
vbYesNo	Oui	Non	
vbYesNoCancel	Oui	Non	Annuler

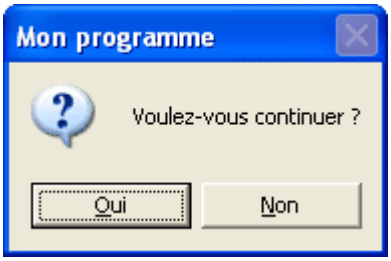
La syntaxe est MsgBox ("Message", attribut bouton):

```
MsgBox ("Voulez-vous continuer ?", vbYesNo)
```



Vous pouvez également y ajouter les icônes et personnaliser le titre de la fenêtre en utilisant la syntaxe : Msgbox ("Message", attribut bouton + attribut icône, "titre de la fenêtre").

```
MsgBox ("Voulez-vous continuer ?", vbYesNo + vbQuestion, _
"Mon programme")
```



MsgBox renvoie une valeur différente pour chaque bouton.

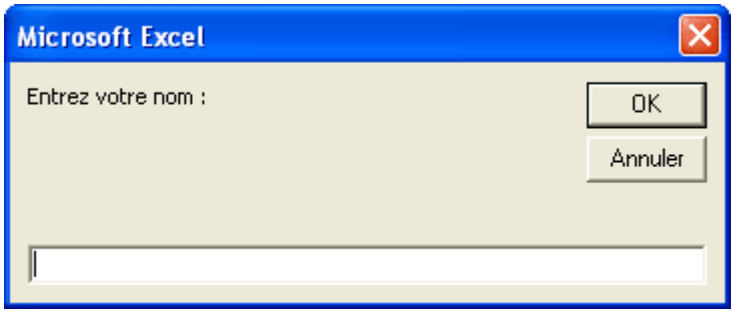
Constante :	*Valeur* :
vbOK	1
vbCancel	2
vbAbort	3
vbRetry	4
vbIgnore	5
vbYes	6
vbNo	7

Ainsi, si l'utilisateur clique sur le bouton "OK", MsgBox renvoie la valeur 1, sur le bouton "Annuler" la valeur 2, sur le bouton "Ignorer" la valeur 5 ...
Cette valeur est récupérée dans une variable.

```
'Dans la ligne d'instruction suivante, si l'utilisateur
'clique sur le bouton "Oui", Reponse prendra comme valeur
'6 sinon Reponse prendra comme valeur 7.
Reponse = MsgBox ("Voulez-vous continuer ?", vbYesNo)
'La ligne suivante arrête la procédure si l'utilisateur
'clique sur "Non"
If Reponse = 7 Then Exit Sub
```

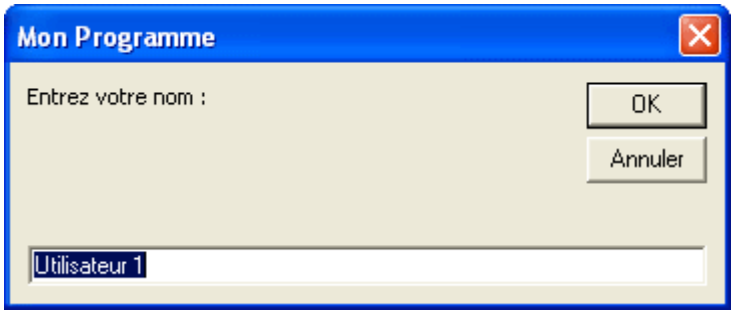
Les InputBox
Les InputBox sont des boites de dialogue dans lesquelles l'utilisateur est invité à entrer des données. La syntaxe est : InputBox ("Message").

```
InputBox ("Entrez votre nom :")
```



Comme pour les MsgBox, vous pouvez changer le titre de la fenêtre. Vous pouvez également entrer une valeur par défaut dans la zone de saisie. La syntaxe devient : InputBox ("Message", "Titre de la fenêtre", "Valeur par défaut").
La valeur saisie peut être récupérée dans une variable. Si l'utilisateur clique sur le bouton "Annuler", la variable renvoie une chaîne de longueur nulle ("").

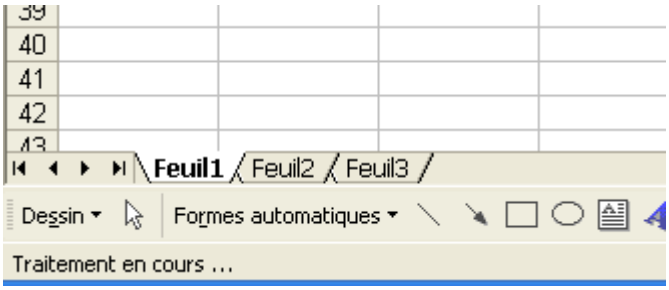
```
Message = InputBox("Entrez votre nom :", "Mon Programme", _
"Utilisateur 1")
```




```
Message = InputBox("Entrez votre nom :", "Mon Programme", _
"Utilisateur 1")
'La ligne suivante arrête la procédure si l'utilisateur
'clique sur "Annuler"
If Message = "" Then Exit Sub
'La ligne suivante place la valeur saisie dans la cellule
'A1 de la feuille active
Range("A1").Value = Message
```

Vous pouvez également écrire un message dans la barre d'état de l'application. La syntaxe est : Application.StatusBar = "Message"

```
Application.StatusBar = "Traitement en cours ..."
```



A la fin de la procédure, pensez à supprimer le message de la barre d'état par la ligne d'instruction: Application.StatusBar = False.

Cours VBA - Les variables -

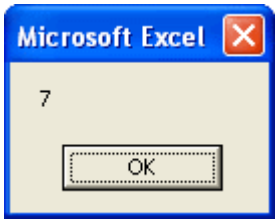
Lors d'une procédure, les variables servent à stocker toutes sortes de données (des valeurs numériques, du texte, des valeurs logiques, des dates ...). Elles peuvent également faire référence à un objet.
Suivant les données que la variable recevra, on lui affectera un type différent. Les différents types de variables de VB sont :

Type de données:	*Mot clé :*	Espace occupé	Plage de valeur
Octet	**Byte**	1 octet	Entier de 0 à 255
Logique	**Boolean**	2 octets	True ou False
Entier	**Integer**	2 octets	Entier de -32 768 à 32 768
Entier Long	**Long**	4 octets	Entier de -2 147 483 648 et 2 147 483 647 à 2 147 483 648 et 2 147 483 647
Décimal simple	**Single**	4 octets	-3,402823E38 à -1,401298E-45 pour les valeurs négatives 1,401298E-45 à 3,402823E38 pour les valeurs positives.
Décimal Double	**Double**	8 octets	-1,79769313486231E308 à -4,94065645841247E-324 pour les valeurs négatives 4,94065645841247E-324 et 1,79769313486231E308 pour les valeurs positives
Monétaire	**Currency**	8 octets	de -922 337 203 685 477,5808 et 922 337 203 685 477,5807
Date	**Date**	8 octets	1er Janvier 100 au 31 décembre 9999
Decimal	**Decimal**	12 octets	+/-79 228 162 514 264 337 593 543 950 335 sans point décimal +/-7,9228162514264337593543950335 avec 28 décimales.
Objet	**Object**	4 octets	toute référence à des objets
Chaîne de caractères à longueur variable	**String**	10 octets + longueur de chaîne	de 0 à 2 milliards de caractères
Chaîne de caractères à longueur fixe	**String**	Longueur de la chaîne	1 à 65 400 caractères

Variant avec chiffres	**Variant**	16 octets	Valeur numérique jusqu'au type double.
Variant avec caractères	**Variant**	22 octets + longueur de la chaîne	Même plage que pour un String de longueur variable
Défini par l'utilisateur	**Type**	Variable	Identique au type de données.

Pour rendre obligatoire la déclaration de variables, placez l'instuction "Option Explicit" sur la première ligne du module ou cochez l'option "Déclaration des variables obligatoires" dans le menu "Outils-Options" de l'éditeur de macros.
La déclaration explicite d'une variable se fait par le mot Dim (abréviation de Dimension). Le nombre maximum de caractères du nom de la variable est de 255. Il ne doit pas commencer par un chiffre et ne doit pas contenir d'espaces. La syntaxe est "Dim NomDeLaVariable as Type".

```
Sub Test()
    Dim SommeVal As Integer
    Dim Val1 As Integer
    Dim Val2 As Integer
    Val1 = 5
    Val2 = 2
    SommeVal = Val1 + Val2
    MsgBox Somme
End Sub
```



Vous pouvez également déclarer vos variables sur une même ligne :

```
Sub Test()
    Dim SommeVal As Integer, Val1 As Integer, Val2 As Integer
    Val1 = 5
    Val2 = 2
    SommeVal = Val1 + Val2
    MsgBox SommeVal
End Sub
```

La portée d'une variable est différente suivant l'endroit et la façon dont elle est déclarée. Une variable déclarée à l'intérieur d'une procédure est dite "Locale". Elle peut-être déclarer par les mots Dim, Static ou Private. Dès que la procédure est terminée, la variable n'est plus chargée en mémoire sauf si elle est déclarée par le mot Static. Une variable Locale est généralement placée juste après la déclaration de la procédure.

```
Option Explicit
'Les variables Val1 et Val2 sont libérées de la mémoire alors
que la variable SommeVal garde sa valeur à la fin de la
procédure
Sub Test()
    Static SommeVal As Integer
    Dim As Val1, Integer, Val2 As Integer
    'Instructions
End Sub
```

Une variable peut être "Locale au module" si celle-ci est déclarée avant la première procédure d'un module. Toutes les procédures du module peuvent alors lui faire appel. Elle est déclarée par les mots Dim ou Private.

```
Option Explicit
'Les variables Val1 et Val2 peuvent être utilisées dans toutes
les procédures du module
Dim As Val1, Integer, Val2 As Integer

Sub Test()
    Static SommeVal As Integer
    SommeVal = Val1 + Val2
End Sub

Sub Test2()
    Static DivisVal As Integer
    DivisVal = Val1 / Val2
End Sub
```

Un variable peut également être accessible à tous les modules d'un projet. On dit alors qu'elle est publique. Elle est déclarée par le mot Public. Elle ne peut pas être déclarée dans un module de Feuille ou dans un module de UserForm.

```
Option Explicit
'Les variables Val1 et Val2 peuvent être utilisées dans toutes
les procédures de tous les modules du projet.
Public As Val1, Integer, Val2 As Integer
```

Une variable peut garder toujours la même valeur lors de l'exécution d'un programme. Dans ce cas, elle est déclarée par les mots Const ou Public Const.

```
Option Explicit
'La variable Chemin gardera toujours la valeur.
Const Chemin as String = "c:\application\excel\"
```

Il est possible de définir une taille fixe pour une variable de type String par la syntaxe Dim Variable as String * Longueur ou Longueur correspond au nombre de caractère que prend la variable.

```
Option Explicit

Sub Test
    Dim Couleur as String * 5
    Couleur = "Rouge"
    ' Si Couleur était égal à "Orange", la variable Couleur
```

```
    aurait pris comme valeur "Orang".
End Sub
```

Il est important de déclarer ses variables par un nom explicite pour rendre le programme plus lisible. Vous pouvez également précéder ce nom par le caractère standard des types de variables. Par exemple, le caractère "i" représente un entier et la variable peut être nommée Dim iNombre as Integer.

Caractère :	*Type de variable* :
b	Boolean
i	Integer
l	long
s	Single
d	Double
c	Currency
dt	Date
obj	Object
str	String
v	Variant
u	Défini par l'utilisateur

Vous pouvez également créer vos propres types de données à l'intérieur du bloc "Type-End Type".

```
Option Explicit
'exemple de création d'un type de données personnalisé
Type Contacts
    Nom As String
    Prenom As String
    Age As Integer
End Type


Sub Test()
    'Déclaration de la variable du type personnalisé
    Dim AjoutContact As Contacts
    AjoutContact.Nom = "TOTO"
    AjoutContact.Prenom = "Titi"
    AjoutContact.Age = 20
End Sub
```

Les variables peuvent également faire référence à des objets comme des cellules, des feuilles de calcul, des graphiques, des classeurs ... Elles sont déclarées de la même façon qu'une variable normale.

```
Option Explicit
```

```
Sub Test()
    'La variable MaCel fait référence à une plage de cellule
    Dim MaCel As Range
    'Le mot Set lui affecte la cellule "A1"
    Set MaCel = Range("A1")
    'La cellule "A1" prend comme valeur 10
    MaCel.Value = 10
End Sub
```

Cours VBA - Classeurs, Feuilles, Cellules -

Les classeurs.
Les classeurs sont désignés par le mot "Workbook". Ils peuvent être ouvert, fermé, enregistré, activé, masqué, supprimé ... par une instruction VB.
Quelques exemples d'instructions sur les classeurs :

```
'Ajouter un nouveau classeur
Workbooks.Add

'Fermer un classeur. Le nom du classeur ou son index peut
être indiqué.
Workbooks("NomDuClasseur.xls").Close

'Fermer le classeur actif.
ActiveWorkbook.Close

'Ouvrir un classeur.
Workbooks.Open "c:\Chemin\NomDuFichier.xls"

'Activer un classeur.
Workbooks("NomDuClasseur.xls").Activate
```

Certaines méthodes de l'objet Workbook possèdent des arguments.
Quelques exemples :

```
'Fermer un classeur sans l'enregistrer
Workbooks("NomDuClasseur.xls").Close False

'Ouvrir un classeur en lecture seule.
Workbooks.Open "c:\Chemin\NomDuFichier.xls", , True

'Enregistrer un classeur sous "Test.xls" avec comme mot
de passe "testpass"
Workbooks(1).SaveAs "test.xls", , "testpass"
```

Les feuilles de calcul.
Les feuilles de calcul sont désignées par le mot "Worksheet". Comme les Workbook, ces objets possèdent de nombreuses propriétés et méthodes.
Quelques exemples d'instructions sur les feuilles :

```
'Selectionner une feuille
Worksheets("Feuil1").Select

'Récupérer le nom de la feuille active dans une variable.
MaFeuille = ActiveSheet.Name

'Masquer une feuille.
Worksheets("Feuil1").Visible = False

'Supprimer une Feuille.
Worksheets("Feuil1").Delete
```

Les exemples précédents font référence aux feuilles du classeur actif. Vous pouvez également faire référence aux feuilles des autres classeurs ouverts :

```
'Copier la Feuil2 de Classeur.xls dans un nouveau
classeur
Workbooks("Classeur.xls").Worsheets("Feuil2").Copy
```

Les cellules.
Une plage de cellules est désignée par l'objet "Range". Pour faire référence à la plage de cellule "A1:B10", on utilisera Range("A1:B10").

```
'Effacer les données et le mise en forme de la plage de
cellule "A1:B10"
Range("A1:B10").Clear
```

L'objet Range permet également de faire référence à plusieurs plages de cellules non contiguës.

```
'Séléctionner les plages de cellule "A1:B5" et "D2:F10"
Range("A1:B5,D2:F10").Select
```

Pour faire référence à une seule cellule, on utilisera l'objet Range("Référence de la cellule) ou Cells(Numéro de ligne, Numéro de colonne).

```
'Ecrire 5 dans la cellule "A3"
Range("A3").Value = 5
'ou
Cells(3, 1).Value = 5
```

Dans l'exemple suivant, nous allons recopier la plage de cellules "A1:B10" de la "Feuil1" du classeur actif dans la cellule "D5" de la "Feuil2" du classeur "Classeur2". Voici à ce que l'enregistreur de macro produirait comme code :

```
Range("A1:B10").Select
Selection.Copy
```

```
Windows("Classeur2").Activate
Sheets("Feuil2").Select
Range("D5").Select
ActiveSheet.Paste
Sheets("Feuil1").Select
Application.CutCopyMode = False
Windows("Classeur1").Activate
```

Voici maintenant le code tel qu'il pourrait être écrit sur une seule ligne de code:

```
Range("A1:B10").Copy Workbooks("Classeur2"). _
Worksheets("Feuil2").Range("D5")
```

On peut utiliser une autre syntaxe pour faire référence à une cellule :

```
'la ligne
Workbooks("Classeur2").Worksheets("Feuil2").Range("D5")
'peut être remplacée par:
Range("[Classeur2]Feuil2!D5")
```

En utilisant des variables objets (très utiles lorsque votre programme fait souvent référence aux mêmes plages de cellules), le code pourrait devenir :

```
Dim Cel1 As Range, Cel2 As Range
Set Cel1 = Range("A1:B1")
Set Cel2 = Workbooks("Classeur2"). _
Worksheets("Feuil3").Range("D5")
Cel1.Copy Cel2
```

VB vous permet également de changer le format des cellules (polices, couleur, encadrement ...). L'exemple suivant applique la police "courrier" en taille 10, en gras, en italique et de couleur rouge. Notez l'utilisation du bloc d'instruction **With - End With** faisant référence à l'objet Font(police) de l'obljet Cel1

```
Dim Cel1 As Range
Set Cel1 = Range("A1")
With Cel1.Font
    .Bold = True
    .Italic = True
    .Name = "Courier"
    .Size = 10
    .Color = RGB(255, 0, 0)
End With
```

A partir d'une cellule de référence, vous pouvez faire appel aux autres cellules par l'instruction "Offset". La syntaxe est Range(Cellule de référence).Offset(Nombre de lignes, Nombre de colonne).

```
'Pour écrire 5 dans la cellule "B2", on pourrait
utiliser :
Range("A1").Offset(1, 1) = 5
'Ecrire une valeur à la suite d'une liste de valeur dans
la colonne A:
Dim NbEnreg As Integer
'NbEnreg correspond au nombre d'enregistrement de la
```

```
colonne A:
NbEnreg = Range("A1").End(xlDown).Row
Range("A1").Offset(NbEnreg, 0) = 10
```

Les arguments (Nombre de lignes, Nombre de colonnes) de l'instruction Offset sont facultatifs et leur valeur par défaut est 0. La dernière ligne de code de l'exemple précédent aurait pu s'écrire :

```
Range("A1").Offset(NbEnreg) = 10
```

Nous verrons l'intérêt de cette instruction dans le cours sur les boucles.

Cours VBA - Classeurs, Feuilles, Cellules -

Les classeurs.
Les classeurs sont désignés par le mot "Workbook". Ils peuvent être ouvert, fermé, enregistré, activé, masqué, supprimé ... par une instruction VB.
Quelques exemples d'instructions sur les classeurs :

```
'Ajouter un nouveau classeur
Workbooks.Add

'Fermer un classeur. Le nom du classeur ou son index peut
être indiqué.
```

```
Workbooks("NomDuClasseur.xls").Close

'Fermer le classeur actif.
ActiveWorkbook.Close

'Ouvrir un classeur.
Workbooks.Open "c:\Chemin\NomDuFichier.xls"

'Activer un classeur.
Workbooks("NomDuClasseur.xls").Activate
```

Certaines méthodes de l'objet Workbook possèdent des arguments.
Quelques exemples :

```
'Fermer un classeur sans l'enregistrer
Workbooks("NomDuClasseur.xls").Close False

'Ouvrir un classeur en lecture seule.
Workbooks.Open "c:\Chemin\NomDuFichier.xls", , True

'Enregistrer un classeur sous "Test.xls" avec comme mot
de passe "testpass"
Workbooks(1).SaveAs "test.xls", , "testpass"
```

Les feuilles de calcul.
Les feuilles de calcul sont désignées par le mot "Worksheet". Comme les Workbook, ces objets possèdent de nombreuses propriétés et méthodes.
Quelques exemples d'instructions sur les feuilles :

```
'Selectionner une feuille
Worksheets("Feuil1").Select

'Récupérer le nom de la feuille active dans une variable.
MaFeuille = ActiveSheet.Name

'Masquer une feuille.
Worksheets("Feuil1").Visible = False

'Supprimer une Feuille.
Worksheets("Feuil1").Delete
```

Les exemples précédents font référence aux feuilles du classeur actif. Vous pouvez également faire référence aux feuilles des autres classeurs ouverts :

```
'Copier la Feuil2 de Classeur.xls dans un nouveau
classeur
Workbooks("Classeur.xls").Worsheets("Feuil2").Copy
```

Les cellules.
Une plage de cellules est désignée par l'objet "Range". Pour faire référence à la plage de cellule "A1:B10", on utilisera Range("A1:B10").

```
'Effacer les données et le mise en forme de la plage de
cellule "A1:B10"
Range("A1:B10").Clear
```

L'objet Range permet également de faire référence à plusieurs plages de cellules non contiguës.

```
'Séléctionner les plages de cellule "A1:B5" et "D2:F10"
Range("A1:B5,D2:F10").Select
```

Pour faire référence à une seule cellule, on utilisera l'objet Range("Référence de la cellule) ou Cells(Numéro de ligne, Numéro de colonne).

```
'Ecrire 5 dans la cellule "A3"
Range("A3").Value = 5
'ou
Cells(3, 1).Value = 5
```

Dans l'exemple suivant, nous allons recopier la plage de cellules "A1:B10" de la "Feuil1" du classeur actif dans la cellule "D5" de la "Feuil2" du classeur "Classeur2". Voici à ce que l'enregistreur de macro produirait comme code :

```
Range("A1:B10").Select
Selection.Copy
Windows("Classeur2").Activate
Sheets("Feuil2").Select
Range("D5").Select
ActiveSheet.Paste
Sheets("Feuil1").Select
Application.CutCopyMode = False
Windows("Classeur1").Activate
```

Voici maintenant le code tel qu'il pourrait être écrit sur une seule ligne de code:

```
Range("A1:B10").Copy Workbooks("Classeur2"). _
Worksheets("Feuil2").Range("D5")
```

On peut utiliser une autre syntaxe pour faire référence à une cellule :

```
'la ligne
Workbooks("Classeur2").Worksheets("Feuil2").Range("D5")
'peut être remplacée par:
Range("[Classeur2]Feuil2!D5")
```

En utilisant des variables objets (très utiles lorsque votre programme fait souvent référence aux mêmes plages de cellules), le code pourrait devenir :

```
Dim Cel1 As Range, Cel2 As Range
Set Cel1 = Range("A1:B1")
Set Cel2 = Workbooks("Classeur2"). _
Worksheets("Feuil3").Range("D5")
Cel1.Copy Cel2
```

VB vous permet également de changer le format des cellules (polices, couleur, encadrement ...). L'exemple suivant applique la police "courrier" en taille 10, en gras, en italique et de couleur rouge. Notez l'utilisation du bloc d'instruction **With - End With** faisant référence à l'objet Font(police) de l'obljet Cel1

```
Dim Cel1 As Range
Set Cel1 = Range("A1")
With Cel1.Font
    .Bold = True
    .Italic = True
    .Name = "Courier"
    .Size = 10
    .Color = RGB(255, 0, 0)
End With
```

A partir d'une cellule de référence, vous pouvez faire appel aux autres cellules par l'instruction "Offset". La syntaxe est Range(Cellule de référence).Offset(Nombre de lignes, Nombre de colonne).

```
'Pour écrire 5 dans la cellule "B2", on pourrait
utiliser :
Range("A1").Offset(1, 1) = 5
'Ecrire une valeur à la suite d'une liste de valeur dans
la colonne A:
Dim NbEnreg As Integer
'NbEnreg correspond au nombre d'enregistrement de la
colonne A:
NbEnreg = Range("A1").End(xlDown).Row
Range("A1").Offset(NbEnreg, 0) = 10
```

Les arguments (Nombre de lignes, Nombre de colonnes) de l'instruction Offset sont facultatifs et leur valeur par défaut est 0. La dernière ligne de code de l'exemple précédent aurait pu s'écrire :

```
Range("A1").Offset(NbEnreg) = 10
```

Nous verrons l'intérêt de cette instruction dans le cours sur les boucles.

Cours VBA - Les tableaux -

Contrairement aux variables classiques qui contiennent une seule valeur, un tableau est une variable qui peut contenir un ensemble de valeurs de même type. Prenons comme exemple notre liste d'élève :

	A	B
1	ELEVE	NOTE
2	PIERRE	5
3	JACQUES	15
4	JEAN	10
5	VALERIE	12
6	PAUL	18
7	DELPHINE	13
8	ANTOINE	0
9	JEANNE	6
10	ANDRE	19
11	JOCELYNE	8
12	FELIX	12
13	JUSTIN	15
14	ETIENNE	6
15	MARIE	5

Pour mettre en mémoire le nom de tous les élèves, déclarons un tableau plutôt que de déclarer autant de variables que d'élèves. La syntaxe pour déclarer un tableau est "**Dim** *Variable*(Nbre éléments) **As** *Type*"

```
'Déclaration du tableau
Dim MesEleves(14) As String
Dim i As Integer
Dim Cel As Range
'On affecte la cellule "A1" à la variable Cel
Set Cel = Range("A1")
'Boucle pour remplir le tableau
For i  = 1 To 14
    MesEleves(i) = Cel.Offset(i)
Next i
```

Dans cet exemple, la variable MesEleves(1) est égale à "PIERRE", MesEleves(2) à "JACQUES",... , MesEleves(15) à "MARIE".

Par défaut, la valeur de l'index inférieur d'un tableau est 1.Pour la changer, on va utiliser le mot **To**. L'exemple suivant va créer un tableau du 5ème au 10 ème élève :

```
'Déclaration du tableau
Dim MesEleves(5 To 10) As String
Dim i As Integer
Dim Cel As Range
```

```
'On affecte la cellule "A1" à la variable Cel
Set Cel = Range("A1")
'Boucle pour remplir le tableau
For i  = 5 To 10
    MesEleves(i) = Cel.Offset(i)
Next i
```

Vous pouvez créer des tableaux contenant des éléments de types différents de deux façons. La première consiste à déclarer la variable de type variant :

```
'Déclaration du tableau
Dim CeJour(4) As Variant
CeJour(1) = 15
CeJour(2) = "Septembre"
CeJour(3) = 2003
CeJour(4) = "Roland"
```

La seconde utilise la fonction **Array**.

```
'Déclaration du tableau
Dim CeJour As Variant
CeJour = Array(15, "Septembre", 2003, "Roland")
```

Dans le 1er cas, CeJour(1) contient 15, CeJour(2) contient "Septembre", CeJour(3) contient 2003 et CeJour(4) contient "Roland".
La valeur du premier index de la fonction Array est 0, donc, dans le second cas, CeJour(0) = 15, CeJour(1) = "Septembre", CeJour(2) = 2003 et CeJour(3) = "Roland".

Vous pouvez créer des tableaux à plusieurs dimensions. Pour mettre en mémoire le nom des élèves avec leurs notes, nous allons créer un tableau à 2 dimensions.

```
'Déclaration du tableau
'14 représente le nombre d'enregistrements
'a traiter, 2 le nombre de champs (Elèves, Notes).
Dim MesEleves(1 To 14, 1 To 2) As Variant
Dim i As Integer
Dim Cel As Range
'On affecte la cellule "A1" à la variable Cel
Set Cel = Range("A1")
'Boucle pour remplir le tableau
For i  = 1 To 14
    'Elèves
    MesEleves(i, 1) = Cel.Offset(i)
    'Notes
    MesEleves(i, 2) = Cel.Offset(i, 1)
Next i
'Ou alors :
Dim j As Integer
For i  = 1 To 14
    For j  = 1 To 2
        MesEleves(i, j) = Cel.Offset(i, j - 1)
    Next j
Next i
```

Dans cet exemple, MesEleves(5, 1) contient "PAUL" et MesEleves(5, 2) la note 18. Notez que la variable a été déclarée de type Variant étant donné qu'elle recevait des données de

type String et des données de type Integer. Si elle était déclarée de type String, les notes seraient en mode texte.

Il est possible de redimensionner un tableau par le mot **Redim**. En effet, le nombre d'éléments ou de dimensions que doit contenir un tableau n'est pas toujours connu. Pour conserver les éléments d'un tableau redimensionné, utilisez le mot **Preserve**.

Dans l'exemple suivant, le tableau va recevoir le nom des élèves dont la note est supérieure ou égale à 10.

```
'Déclaration du tableau
Dim MesEleves() As String
Dim i As Integer
Dim j As Integer 'Nbre éléments du tableau
Dim Cel As Range
'On affecte la cellule "A1" à la variable Cel
Set Cel = Range("A1")
'Boucle pour remplir le tableau
For i  = 1 To 14
    If Cel.Offset(i, 1) >= 10 Then 'Si la note >=10
        j = j + 1
        'Redimension du tableau en conservant
        'ses éléments
        ReDim Preserve MesEleves(j)
        MesEleves(j) = Cel.Offset(i)
    End If
Next i
```

Le tableau contient 8 éléments et, par exemple, la valeur de MesEleves(5) est "DELPHINE".

Cours VBA - Les Fonctions -

Les fonctions sont des procédures qui renvoient une valeur. Elles peuvent posséder des arguments qui représentent des variables définies dans l'appel de la fonction. Dans l'exemple suivant, une fonction calculant la somme de deux entiers est appelée d'une procédure.

```
'Déclaration de la fonction de type Integer
Function MaSomme(Nbre1 As Integer, Nbre2 As Integer)As Integer
    MaSomme = Nbre1 + Nbre2
End Function

'Procédure appelant la fonction
Sub Calcul ()
    Dim Resultat As Integer
    Dim Val1 As Integer, Val2 As Integer
    Val1 = 2
    Val2 = 3
    'Appel de la fonction avec ses arguments
    Resultat = MaSomme(Val1, Val2) 'Resultat = 5
End Sub
```

Reprenons le tableau des élèves ainsi que la correspondance des mentions par rapport aux notes.

	A	B
1	ELEVE	NOTE
2	PIERRE	5
3	JACQUES	15
4	JEAN	10
5	VALERIE	12
6	PAUL	18
7	DELPHINE	13
8	ANTOINE	0
9	JEANNE	6
10	ANDRE	19
11	JOCELYNE	8
12	FELIX	12
13	JUSTIN	15
14	ETIENNE	6
15	MARIE	5

Notes :	*Mention* :
0	Nul
1 à 5	Moyen
6 à 10	Passable
11 à 15	Bien
16 à 19	Très bien

La fonction suivante va définir la mention par rapport aux notes.

```
'Déclaration de la fonction de type String
Function Mention(Note As Integer)As String
    Select Case Note
        Case 0
            Mention = "Nul"
        Case 1 To 5
            Mention = "Moyen"
        Case 6 To 10
            Mention = "Passable"
        Case 11 To 15
            Mention = "Bien"
        Case 16 To 19
            Mention = "Très Bien"
        Case Else
            Mention = "Excellent"
    End Select
End Function
```

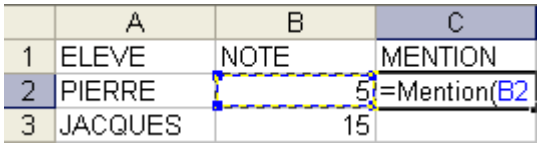
La procédure suivante va appeler la fonction "Mention" pour chaque élève et inscrire sa valeur dans la colonne C.

```
Sub Calcul_Mention ()
    Dim i As Integer 'Nbre d'élève
    Dim ValNote As Integer
    For i = 1 To 14
        ValNote = Range("A1").Offset(i, 1)
        'L'instruction suivante va placer dans la colonne C
        'la valeur de la fonction "Mention"
        Range("C1").Offset(i) = Mention(ValNote)
    Next i
End Sub
```

	A	B	C
1	ELEVE	NOTE	MENTION
2	PIERRE	5	Moyen
3	JACQUES	15	Bien
4	JEAN	10	Passable
5	VALERIE	12	Bien
6	PAUL	18	Très Bien
7	DELPHINE	13	Bien
8	ANTOINE	0	Nul
9	JEANNE	6	Passable
10	ANDRE	19	Très Bien
11	JOCELYNE	8	Passable
12	FELIX	12	Bien
13	JUSTIN	15	Bien
14	ETIENNE	6	Passable
15	MARIE	5	Moyen

Les fonctions peuvent également être appelées d'une feuille de calcul comme toutes les fonctions intégrées d'Excel. Dans notre exemple, on aurait pu saisir "=Mention(B2)" dans la

cellule "C2" puis recopier cette formule sur la plage "C3:C15" pour obtenir le même résultat. En saisissant directement la fonction dans la feuille de calcul, la valeur de la mention change si la note est modifiée.

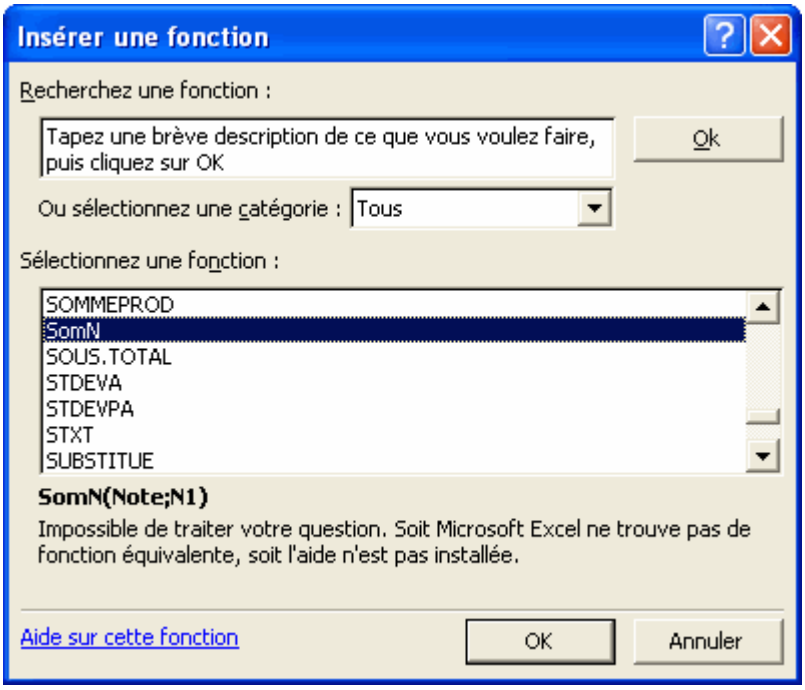


	A	B	C
1	ELEVE	NOTE	MENTION
2	PIERRE	5	=Mention(B2
3	JACQUES	15	

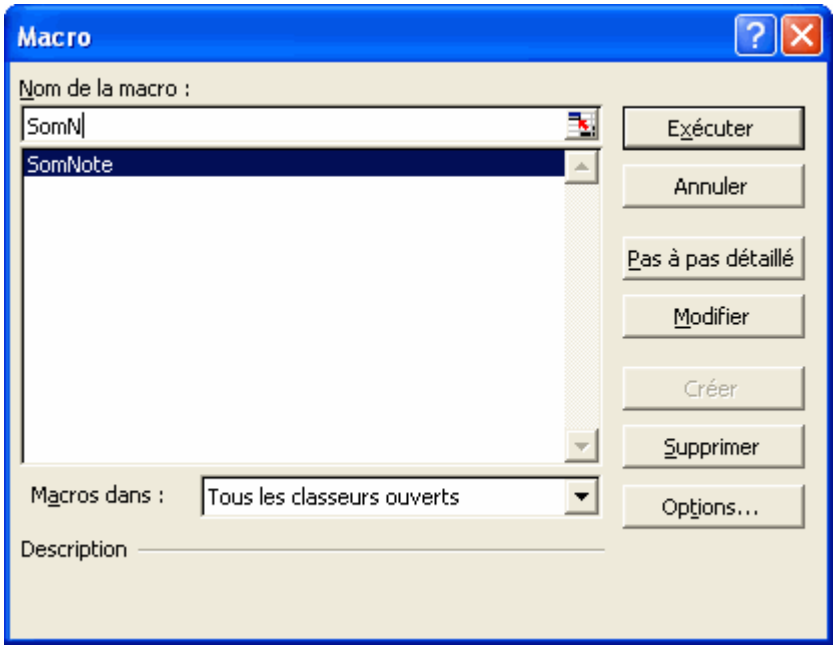
Dans les exemples précédents, les arguments des fonctions sont obligatoires. Pour rendre un argument facultatif, il faut le déclarer par le mot Optional. La fonction suivant va calculer la somme des notes avec comme option, une note minimale.

```
'Déclaration de la fonction de type Integer
Function SomN(Note As Range, Optional N1 As Integer)As Integer
    Dim Item As Variant
    'Item prend la valeur de chaque cellule de la plage de
    'cellule définie dans l'argument Note
    For Each Item In Note
        'si l'argument N1 est défini
        If N1 > 0 Then
            'test si la valeur de la cellule est > ou = à N1
            If Item >= N1 Then
                SomN = SomN + Item
            End If
        Else 'l'argument N1 n'est pas défini ou est = 0
            SomN = SomN + Item
        End If
    Next Item
End Function

'Appel de la fonction sans argument N1
Sub SommeNote ()
    Dim Resultat As Integer
    Resultat = SomN(Range("B2:B15")) 'Resultat = 144
End Sub

'Appel de la fonction avec argument N1
Sub SommeNote ()
    Dim Resultat As Integer
    Resultat = SomN(Range("B2:B15"), 10) 'Resultat = 114
End Sub
```

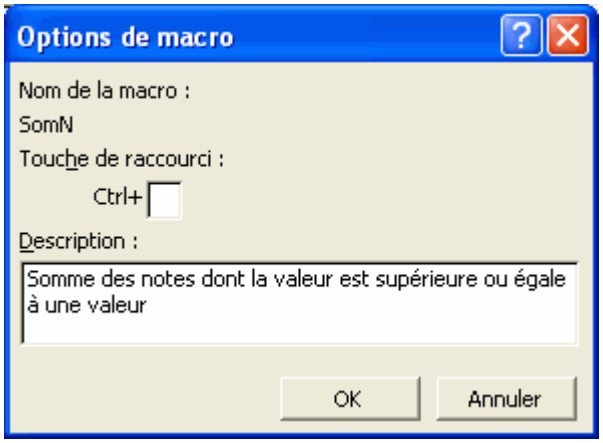
Vos fonctions apparaissent dans la liste des fonctions d'Excel accessible par le menu "Insertion-Fonctions".



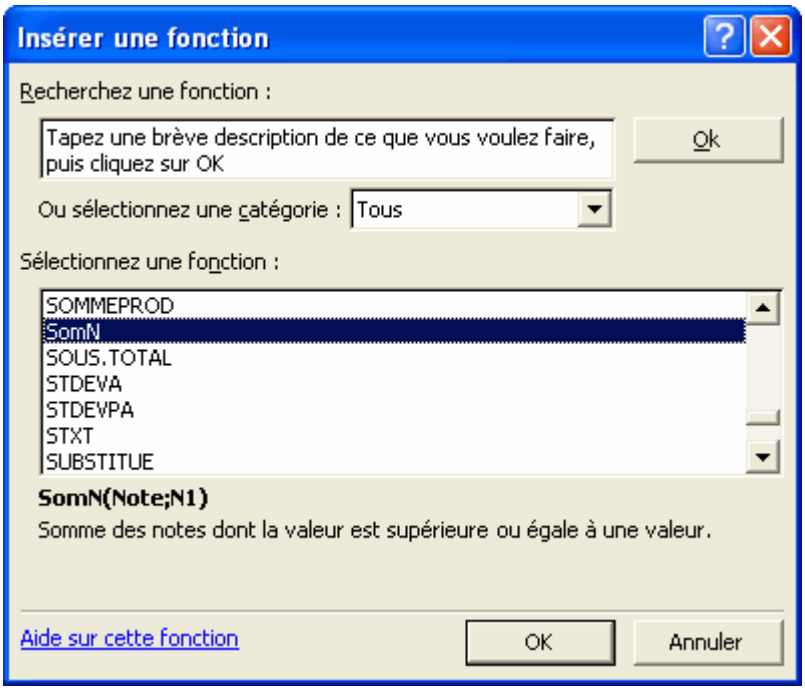
Pour ajouter une zone d'aide à vos fonctions, ouvrez la boite de dialogue Macro par le menu "Outils-Macro-Macros". Dans cette boite, seuls les procédures apparaissent. Saisissez le nom de votre procédure puis cliquez sur "Options".



Saisissez votre texte dans la zone description puis validez.



Voila comment apparaît votre fonction dans la liste des fonctions :



Cours VBA - Fonctions de texte -

VBA possède des fonctions permettant d'extraire une chaîne de caractères d'un texte.
La fonction **Len** renvoie le nombre de caractères d'un texte.

```
Dim Message As String,  Longueur As Integer
Message =  "Fonctions de texte"
Longueur = Len(Message)     'Longueur renvoie 18
```

La fonction **Left** renvoie un nombre de caractères en partant de la gauche. La syntaxe est Left(Texte, Nombre de caractères).

```
Dim Message As String, MTexte As String
Message =  "Fonctions de texte"
MTexte = Left(Message, 1)     'Renvoie "F"
MTexte = Left(Message, 9)     'Renvoie "Fonctions"
```

La fonction **Right** renvoie un nombre de caractères en partant de la droite. La syntaxe est Right(Texte, Nombre de caractères).

```
Dim Message As String, MTexte As String
Message =  "Fonctions de texte"
MTexte = Right(Message, 1)     'Renvoie "e"
MTexte = Right(Message, 8)     'Renvoie "de texte"
```

La fonction **Mid** renvoie un nombre de caractères en partant d'un caractère défini. La syntaxe est Mid(Texte, Départ, Nombre de caractères). Si le Nombre de caractères n'est pas indiqué, la fonction renvoie tous les caractères à partir de la position départ.

```
Dim Message As String, MTexte As String
Message =  "Fonctions de texte"
MTexte = Mid(Message, 2, 5)     'Renvoie "oncti"
MTexte = Mid(Message, 11, 2)     'Renvoie "de"
MTexte = Mid(Message, 11)     'Renvoie "de texte"
```

La fonction **LTrim** supprime les espaces se trouvant avant la chaîne de caractères.

```
Dim Message As String, MTexte As String
Message =  "   Fonctions   "
MTexte = LTrim(Message)     'Renvoie "Fonctions   "
```

La fonction **RTrim** supprime les espaces se trouvant après la chaîne de caractères.

```
Dim Message As String, MTexte As String
Message =  "   Fonctions   "
MTexte = RTrim(Message)     'Renvoie "   Fonctions"
```

La fonction **Trim** supprime les espaces se trouvant avant et après la chaîne de caractères.

```
Dim Message As String, MTexte As String
Message =  "   Fonctions   "
MTexte = Trim(Message)     'Renvoie "Fonctions"
```

La fonction **Ucase** convertie le texte en majuscules.

```
Dim Message As String, MTexte As String
Message =  "Fonctions de texte"
MTexte = Ucase(Message)
'Renvoie "FONCTIONS DE TEXTE"
```

La fonction **Lcase** convertie le texte en minuscules.

```
Dim Message As String, MTexte As String
Message =  "Fonctions de texte"
MTexte = Lcase(Message)
'Renvoie "fonctions de texte"
```

La fonction **Application.Proper** convertie le texte en nom propre.

```
Dim Message As String, MTexte As String
Message =  "Fonctions de texte"
MTexte = Application.Proper(Message)
'Renvoie "Fonctions De Texte"
```

La fonction **Replace** permet de remplacer une chaîne de caractères par une autre.

```
Dim Message As String, MTexte As String
Message =  "Fonctions de texte"
MTexte = Replace(Message, "te", "et")
'Renvoie "Fonctions de etxet"
```

Cette fonction possède des arguments facultatifs permettant de déterminer la position du premier remplacement et le nombre de remplacement à effectuer. La syntaxe est **Replace**(Texte, Chaine à remplacer, chaîne de remplacement, Départ, Nombre de remplacement).

```
Dim Message As String, MTexte As String
Message =  "Fonctions de texte"
MTexte = Replace(Message, "t", "WW", 3, 1)
'Renvoie "ncWWions De texte"

MTexte = Replace(Message, "t", "WW", , 2)
'Renvoie "FoncWWions de WWexte"
```

La fonction **Val** renvoie la valeur numérique d'une chaîne de caractères. Si la chaîne de caractères est composée de chiffres et de lettres, la valeur s'arrête au premier caractère non numérique.

```
Dim Message As String, MTexte As Double
Message =  "2003"
MTexte = Val(Message)
'Renvoie 2003

Message =  "a 2003"
MTexte = Val(Message)
'Renvoie 0

Message =  "   2003 2004"
MTexte = Val(Message)
'Renvoie 20032004

Message =  "2003 et 2004"
MTexte = Val(Message)
'Renvoie 2003
```

La fonction **IsNumeric** permet de tester si une chaîne de caractères est numérique. Elle renvoie une valeur de type Boolean.

```
Dim Message As String, MTexte As Integer
Message =  100

If IsNumeric(Message) = True Then
    MTexte = Message + 10 'MTexte prend la valeur 110
End If
```

La fonction **IsDate** permet de tester si une chaîne de caractères est une date. Elle renvoie une valeur de type Boolean.

```
Dim Message As String, MTexte As Integer
Message =  "1 MARS 2000"

If IsDate(Message) = True Then
    MTexte = Month(Message) 'MTexte prend la valeur
                             3(3ème mois de l'année)
End If
```

Certaines fonctions permettent de convertir des données en d'un type défini. Par exemple, la fonction **CDate** va convertir des données en date.
Tableau de fonctions de conversions de données :

Fonctions :	*Type* :
CBool	Boolean
CByte	Byte
CCur	Currency
CDate	Date
CDbl	Double
CDec	Decimal
CInt	Integer
CLng	Long
CSng	Single
CStr	String
CVar	Variant

```
Dim Message As Double, MTexte As Integer
Message =  325.25
MTexte = CInt(Message) 'MTexte prend la valeur 325
```

Le format des dates et heures est défini par la fonction **Format**. La syntaxe est **Format**(MaDate, Format).

```
Dim MaDate As Date, MDate As String
MaDate =  date 'date du jour
MDate = Format(Message, "dd mmmm yyyy") 'MDate prend la
valeur "05 Octobre 2003"
```

La fonction **Format** permet également de formater les nombres.

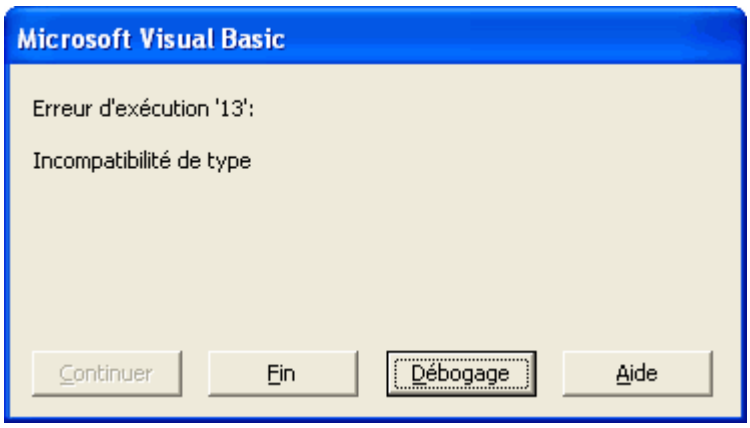
```
Dim MonNombre As String
MonNombre = Format(1500, "0 000") 'MonNombre prend la
'valeur "1 500"
MonNombre = Format(1500, "0 000.00 Euros") 'MonNombre
'prend la valeur "1 500.00 Euros"
```


Cours VBA - Gestion des erreurs - Déboguage -

L'instruction On Error permet la gestion des erreurs lors de procédures.

```
Sub Test()
    Dim i As Integer
    i = "coucou"
    Suite des instructions
End Sub
```

Cette procédure s'arrêtera et un message d'erreur apparaîtra car la variable i est déclarée de type Integer.



L'instruction On Error Resume Next permet de passer sur les erreurs et continue la procédure à la ligne d'instruction suivante. La procédure suivante ne s'arrêtera pas.

```
Sub Test()
    Dim i As Integer
    On Error Resume Next
```

```
    i = "coucou"
    Suite des instructions
End Sub
```

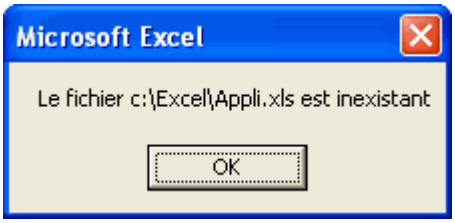
L'instruction On Error GoTo 0 invalide l'instruction On Error Resume Next. Dans la procédure suivante, les erreurs de *Instructions 1* seront ignorées alors que la procédure s'arrêtera sur les erreurs de *Instructions 2*.

```
Sub Test()
    On Error Resume Next
    instructions1
  On Error GoTo 0
    instructions2
End Sub
```

L'instruction On Error Goto *Etiquette* continue la procédure à une étiquette définie dans cette même procédure.

```
Sub Test()
    Dim Fichier As String
    Fichier = "c:\Excel\Appli.xls"
    On Error GoTo MsgErreurs
    Workbooks.open Fichier
  instructions
  Exit Sub 'Arrête la procédure pour éviter le message
MsgErreurs :
    Msgbox "Le fichier " & Fichier & " est inexistant"
End Sub
```

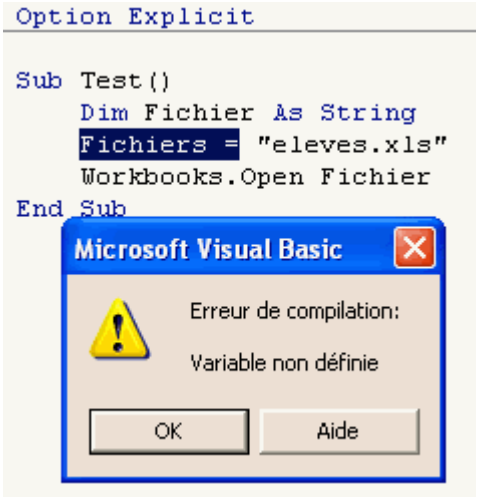
Si le fichier "c:\Excel\Appli.xls" n'existent pas, la procédure se rend à l'étiquette MsgErreurs, affiche le message puis s'arrête.



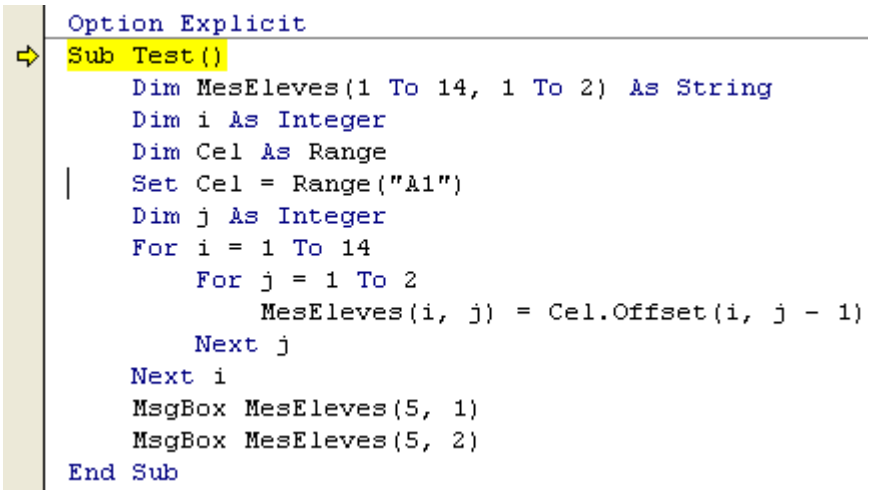
La procédure suivante réalise la même chose mais, après le message reprend, à la ligne qui suit la ligne qui a provoquée l'erreur.

```
Sub Test()
    Dim Fichier As String
    Fichier = "c:\Excel\Appli.xls"
    On Error GoTo MsgErreurs
    Workbooks.open Fichier
    On Error GoTo 0 'Invalide la gestion d'erreurs
  instructions
  Exit Sub 'Arrête la procédure pour éviter le message
MsgErreurs :
    Msgbox "Le fichier " & Fichier & " est inexistant"
    Resume Next
End Sub
```

VBA possède des outils permettent de tester votre application.
Compilez votre projet par le menu "Débogage-Compiler VBAProject". Cette opération vous affichera les erreurs de votre code tels des variables non ou mal déclarées.



Testez votre application "Pas à Pas" en cliquant dans la procédure puis faîtes "F8". La ligne d'instruction lue est alors surlignée et une flèche s'affiche à sa hauteur dans la marge. A chaque touche "F8", la procédure avance d'une ligne.



Vous pouvez à tout moment déplacer la flèche pour vous déplacer dans la procédure.

```
Option Explicit
Sub Test()
    Dim MesEleves(1 To 14, 1 To 2) As String
    Dim i As Integer
    Dim Cel As Range
    Set Cel = Range("A1")
    Dim j As Integer
    For i = 1 To 14
        For j = 1 To 2
            MesEleves(i, j) = Cel.Offset(i, j - 1)
        Next j
    Next i
    MsgBox MesEleves(5, 1)
    MsgBox MesEleves(5, 2)
End Sub
```

Placez des points d'arrêts pour arrêter votre procédure ou vous le souhaitez. Pour cela, cliquez dans la marge à la hauteur de la ligne désirée. Vous pouvez créer autant de points d'arrêt que nécessaire.

```
Option Explicit
Sub Test()
    Dim MesEleves(1 To 14, 1 To 2) As String
    Dim i As Integer
    Dim Cel As Range
    Set Cel = Range("A1")
    Dim j As Integer
    For i = 1 To 14
        For j = 1 To 2
            MesEleves(i, j) = Cel.Offset(i, j - 1)
        Next j
    Next i
    MsgBox MesEleves(5, 1)
    MsgBox MesEleves(5, 2)
End Sub
```

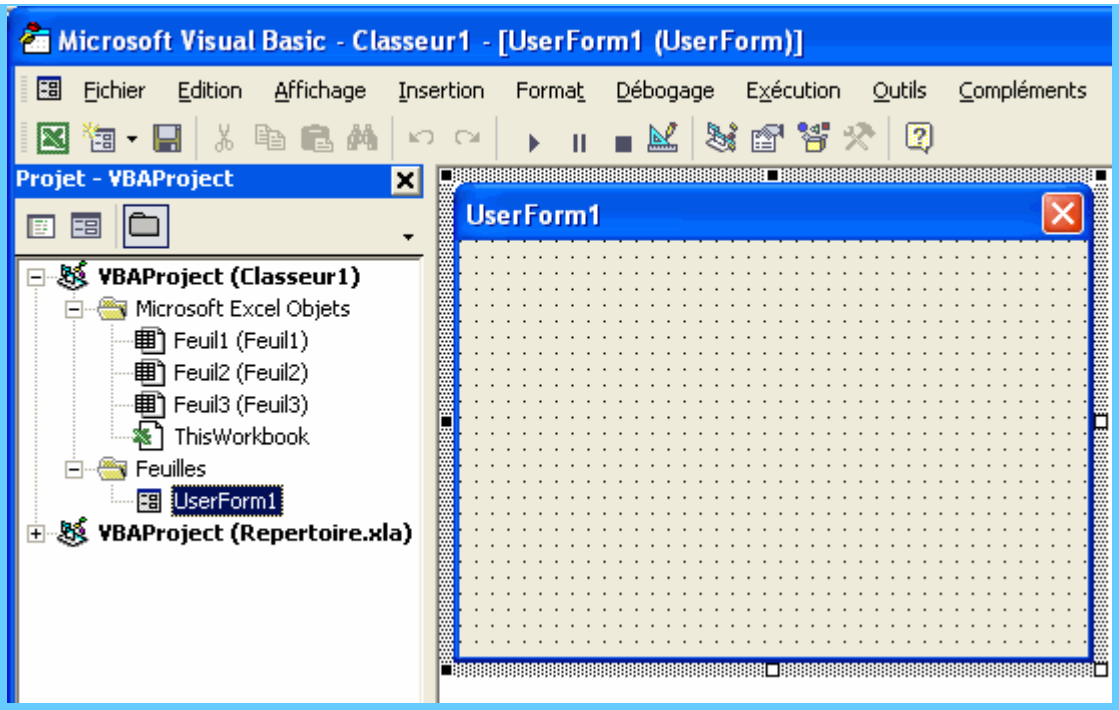
Contrôlez la valeur de vos variables en les survolant avec la souris.

```
Option Explicit
Sub Test()
    Dim MesEleves(1 To 14, 1 To 2) As String
    Dim i As Integer
    Dim Cel As Range
    Set Cel = Range("A1")
    Dim j As Integer
    For i = 1 To 14
        For j = 1 To 2
            MesEleves(i, j) = Cel.Offset(i, j - 1)
        Next j
    Next i
    MsgBox MesEleves(5, 1)
    MsgB MesEleves(5, 1) = "PAUL" 2)
End Sub
```

Site Classé

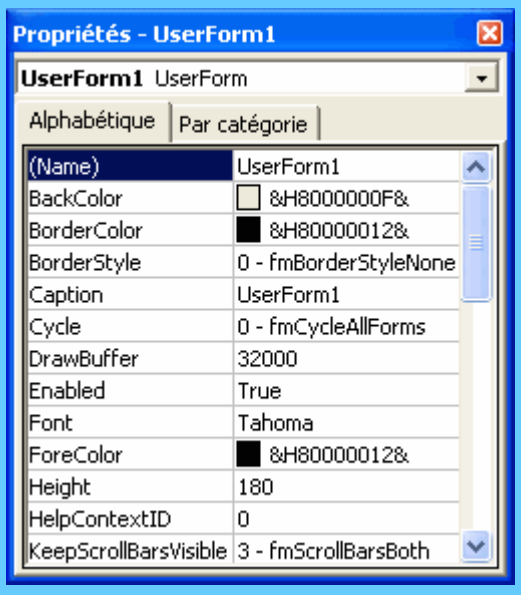
Cours VBA - UserForm - Présentation -

Les UserForm sont des boites de dialogues personnalisées, offrant une interface intuitive entre l'application et l'utilisateur.
Sous VBE, les UserForm sont créés par le menu "Insertion-UserForm".

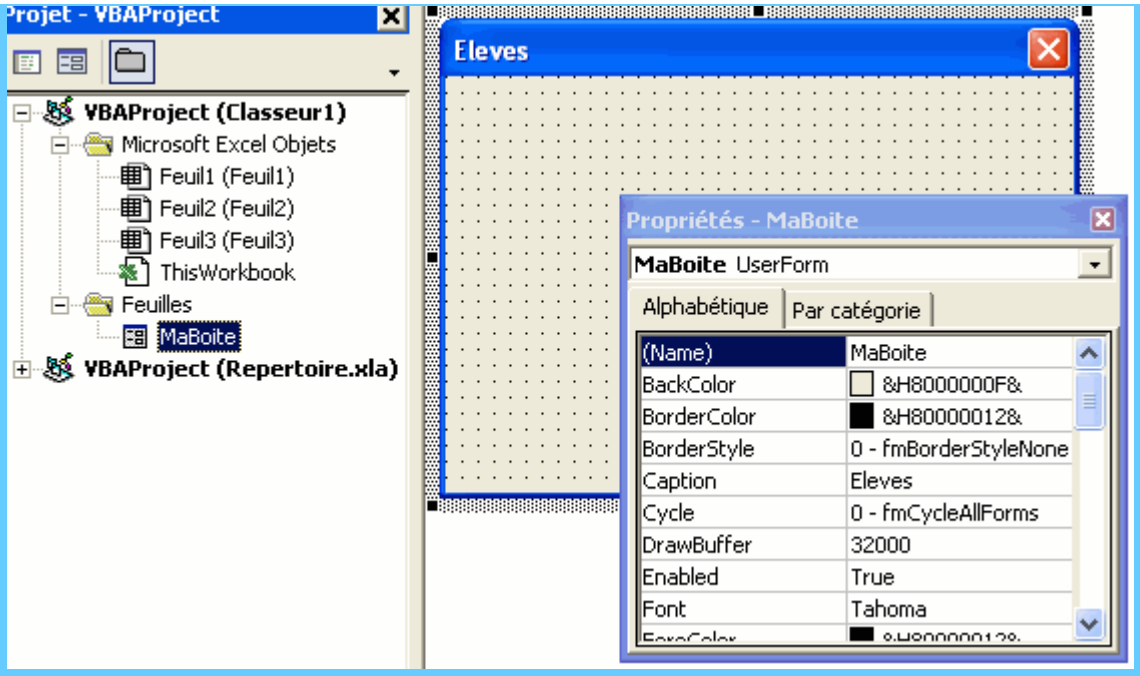


Par défaut, les UserForm sont nommés "UserForm1", "UserForm2" ...
Chaque UserForm possède ses propres propriétés tel que son nom, ses couleurs, sa taille, sa position ...

Les propriétés d'un UserForm s'affichent en cliquant sur l'icône  , par le menu "Affichage-Fenêtre Propriétés" ou par la touche "F4".

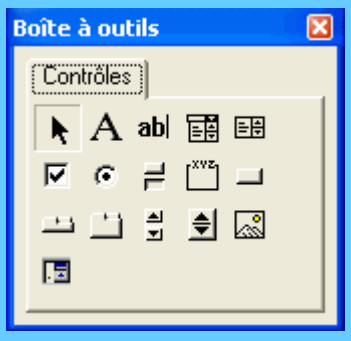


La propriété "Name" change le nom de l'UserForm, la propriété "Caption", son titre.

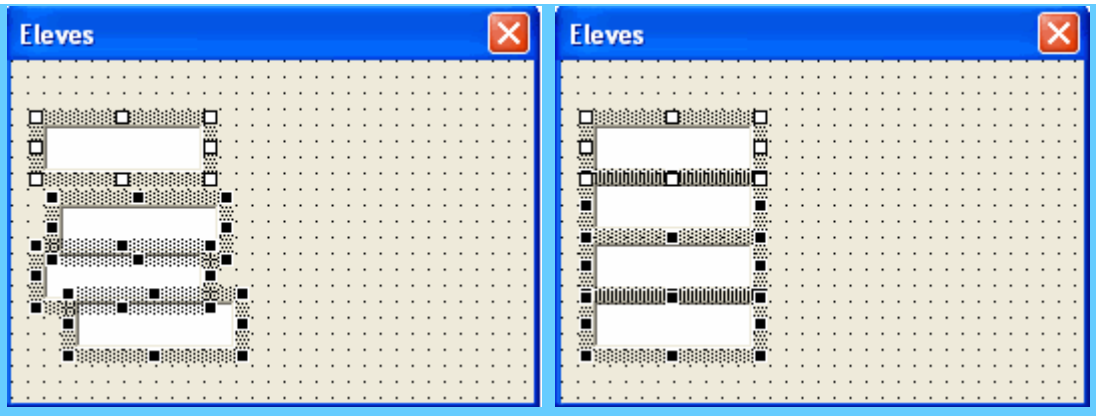


Les propriétés permettent de personnaliser les UserForm. Vous pouvez changer la couleur de fond par la propriété "BackColor", ajouter une bordure par la propriété "BorderStyle", définir sa couleur par la propriété "BorderColor", mettre une image de fond par la propriété "Picture" ...
Le dimensionnement d'un UserForm peut se faire avec la souris ou en définissant sa taille par ses propriétés "Width" (Largeur) et "Height" (Hauteur).

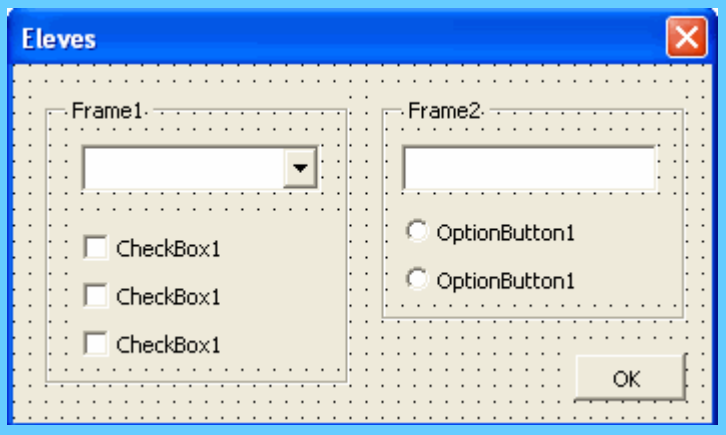
Chaque UserForm va recevoir des contrôles. En cliquant sur le UserForm, une boite à outils doit apparaître. Si ce n'est pas le cas, affichez la en cliquant sur l'icône ou par le menu "Affichage-Boîte à outils".



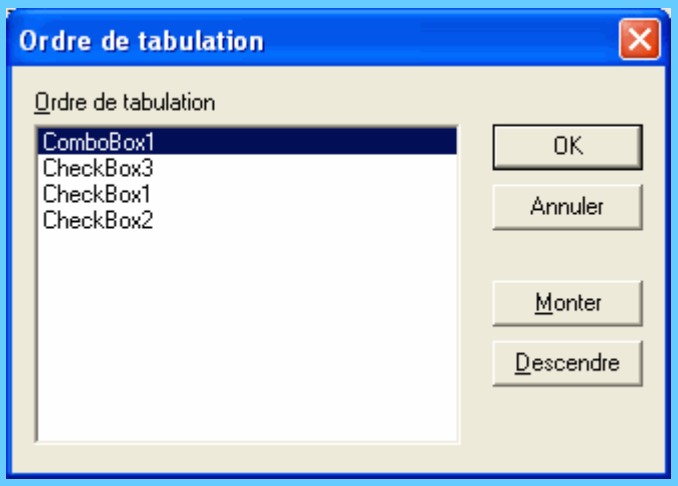
Pour ajouter un contrôle sur le UserForm, vous pouvez soit cliquer sur le contrôle désiré puis, sur le UserForm, tracer un rectangle qui définira sa taille ou simplement faire un cliquer-glisser du contrôle sur l'UserForm.
Les UserForm possèdent une grille matérialisée par des points. Elle permet l'alignement des contrôles. Vous pouvez la masquer, la désactiver ou définir sa taille par le menu "Outils-Options" dans l'onglet "Général".
Le menu "Fornnnnnnnnnnmat" de VBE permet d'aligner les contrôles.
Par exemple le menu "Format-Aligner- Gauche" puis le menu "Espacement Vertical-Egaliser" permet un alignement régulier des contrôles:



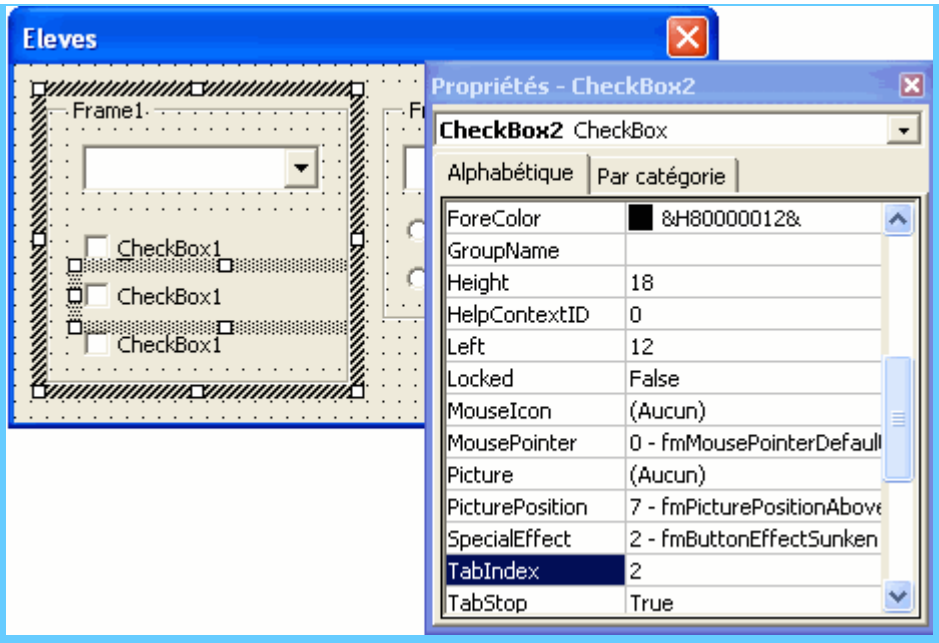
Le contrôle "Frame"  permet de grouper des contrôles.



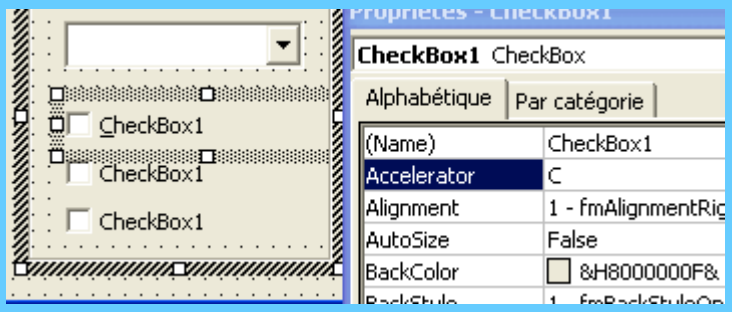
Le UserForm doit permettre à l'utilisateur de passer d'un contrôle à l'autre par la touche "Tabulation" de façon ordrée. Le menu "Affichage-Ordre de tabulation" permet de paramétrer l'ordre de tabulation. Cliquez sur l'UserForm pour changer l'ordre des deux frames et du bouton "OK" et sélectionnez une frame pour changer l'ordre des contrôles qu'elle contient.



Vous pouvez également changer l'ordre de tabulation par la propriété "TabIndex" de chaque contrôle.

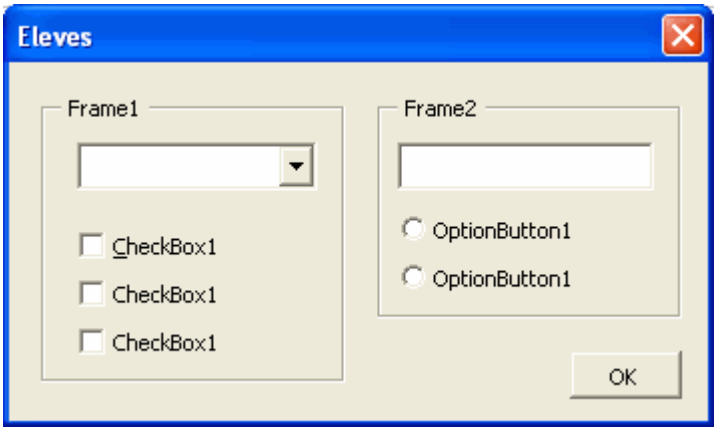


Vous pouvez affecter une touche de raccourci "Alt+caractère" à un contrôle par sa propriété "Accelerator". Utilisez un caractère du nom du contrôle, celui-ci sera souligné, indiquant à l'utilisateur quelle touche utiliser :



L'affichage des UserForm s'effectue par la méthode "Show" de l'UserForm. Cette instruction doit être placée à l'intérieur d'une procédure dans un module.

```
Sub AfficheUF()
    MaBoite.Show
End Sub
```

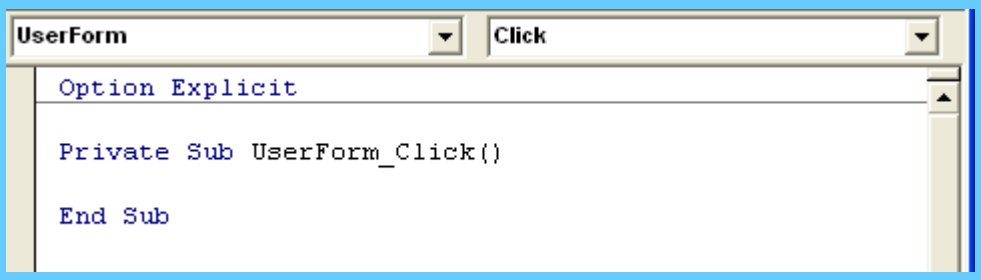


Par défaut, un UserForm est modal, c'est à dire que l'utilisateur ne peut effectuer aucune action sur l'application tant qu'il n'est pas fermé. Depuis la version 2000 d'Excel, il est possible d'afficher des boites non modal, permettant l'utilisation des feuilles de calcul en gardant le UserForm affichée. La syntaxe est :

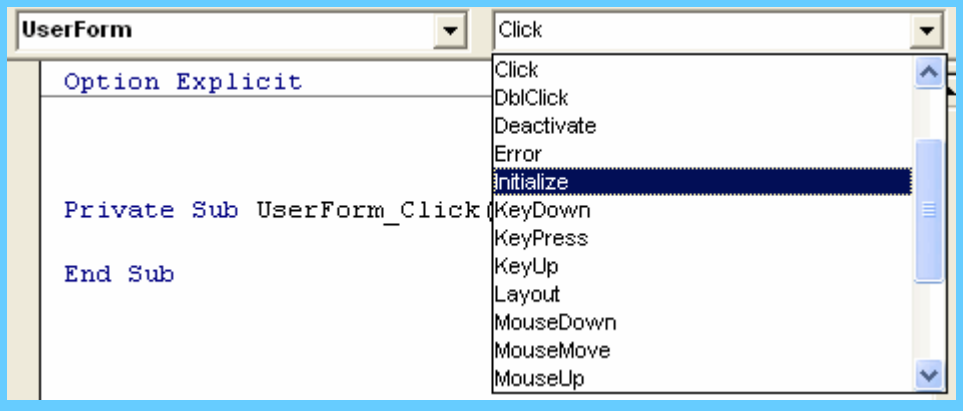
```
Sub AfficheUF()
    MaBoite.Show 0
End Sub
```

L'instruction Load charge le UserForm en mémoire sans l'afficher.
L'instruction Unload ferme le UserForm en le déchargeant de la mémoire. La syntaxe de cette instruction est : Unload UserForm.
Il est également possible de fermer un UserForm en gardant en mémoire la valeur de ses contrôles par la méthode Hide. La syntaxe devient : UserForm.Hide.

Chaque UserForm possède son propre module.
Pour y accéder, cliquez sur le UserForm ou sur un contrôle puis tapez "F7" ou faîtes un double-clic sur l'objet. Par défaut, le module s'affichera avec une procédure évènementielle de type privée de l'objet sélectionné.

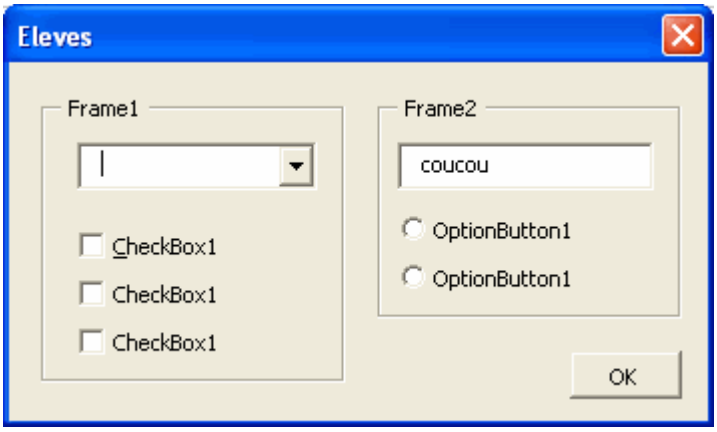


Les deux listes déroulantes en haut du module permettent de sélectionner l'objet et son évènement.



Dans cet exemple, la procédure Initialize de l'objet UserForm va être créée et ses instructions vont être exécutées au chargement de la boite.

```
Private Sub UserForm_Initialize()
    TextBox1 = "coucou"
End Sub
```



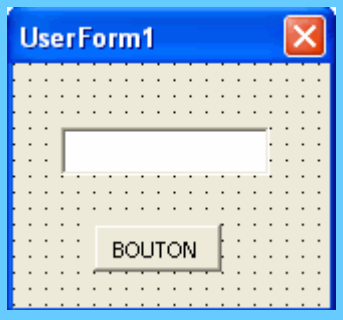
Si l'évènement Initialize se produit au chargement d'un UserForm, l'évènement QueryClose se produit à sa fermeture. Dans l'exemple suivant, un message contenant le texte de l'objet TextBox1 s'affichera à la fermeture de la boite.

```
Sub UserForm_QueryClose(Cancel As Integer, CloseMode As Integer)
    Msgbox TextBox1
End Sub
```

L'élément Cancel de l'évènement QueryClose invalide la fermeture de la boite si sa valeur est 1 et l'élément CloseMode défini la manière dont la boite cherche à être fermée. Si l'utilisateur cherche à la fermer en cliquant sur la croix, CloseMode prend comme valeur 0, sinon CloseMode prend comme valeur 1. L'exemple suivant montre comment obliger l'utilisateur à fermer la boite en cliquant sur le bouton "OK".

```
Private Sub CommandButton1_Click()
    Unload MaBoite
End Sub

PrivateSub UserForm_QueryClose(Cancel As Integer, CloseMode As Integer)
    If CloseMode = 0 Then 'Si on clique sur la croix
        Msgbox "Fermez la boite avec le bouton OK"
        Cancel = 1 'Invalide la fermeture
    End If
End Sub
```

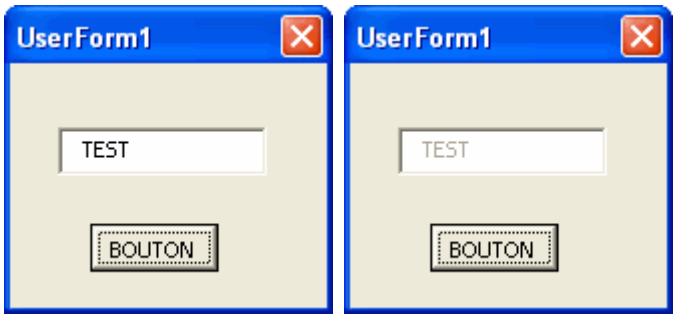
Les contrôles possèdent la propriété Visible qui permet de les rendre visible ou invisible. Dans l'exemple suivant, le click sur BOUTON va masquer ou afficher la zone de texte.



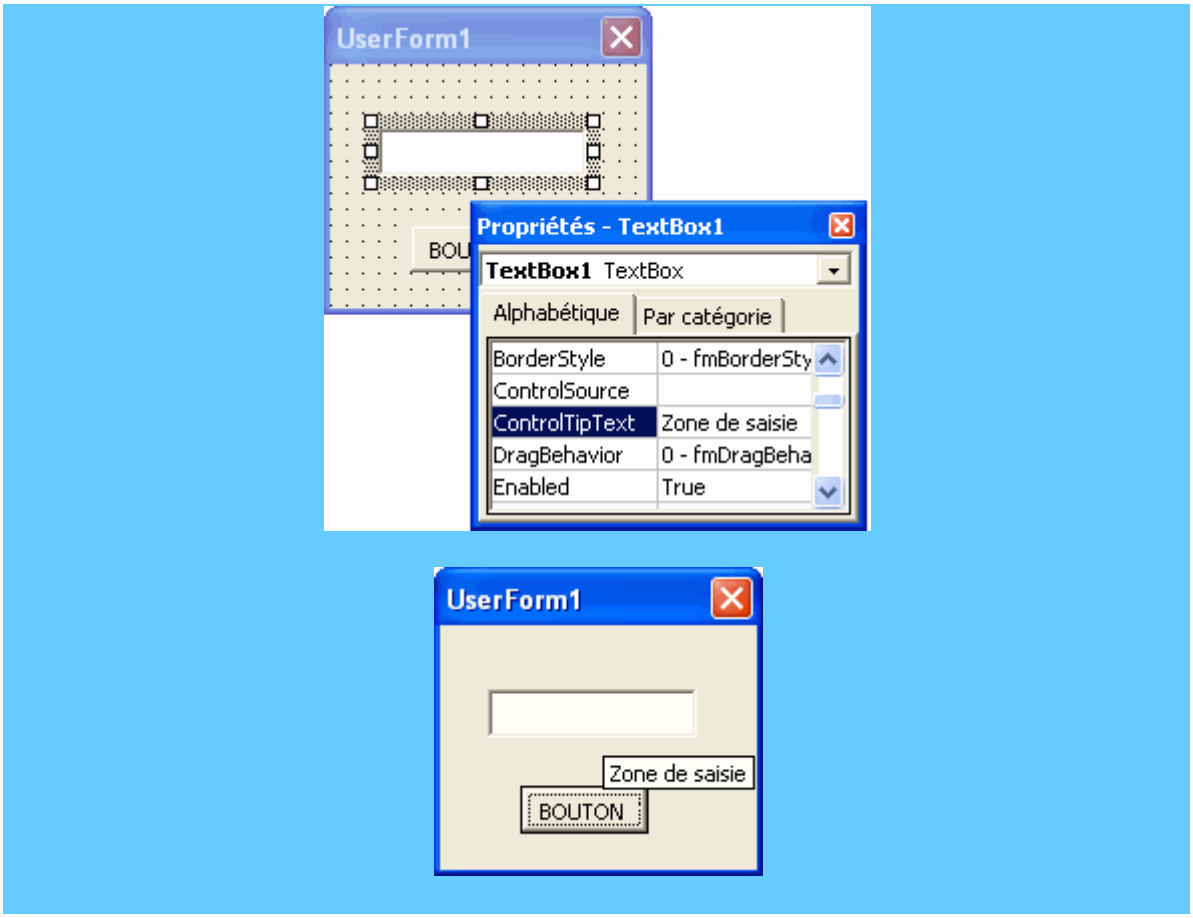
```
Private Sub CommandButton1_Click()
    If TextBox1.Visible = True Then
        TextBox1.Visible = False
    Else
        TextBox1.Visible = True
    End If
End Sub
```

Les contrôles possèdent la propriété Enabled qui peut interdire son accès à l'utilisateur. Lorsqu'on contrôle n'est pas accessible, son aspect change.
Dans l'exemple suivant, le click sur BOUTON va interdire ou rendre accessible l'accès à la zone de texte.

```
Private Sub CommandButton1_Click()
    If TextBox1.Enabled = True Then
        TextBox1.Enabled = False
    Else
        TextBox1.Enabled = True
    End If
End Sub
```



Les contrôles possèdent la propriété ControlTipText qui affiche une étiquette lors du survol de la souris.



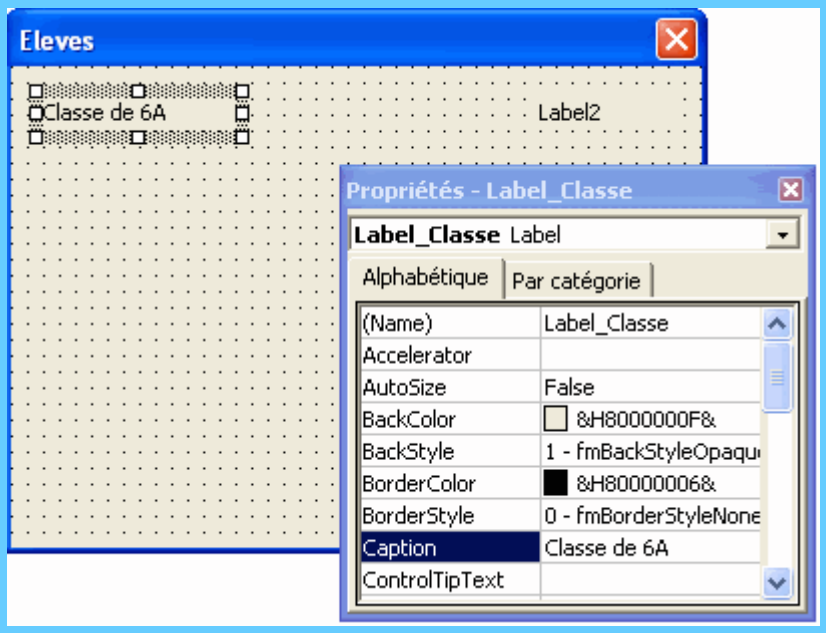
Site Classé

Cours VBA - UserForm - Les contrôles -

La boite à outils affiche les contrôles standard de VBA.

Sélection
Cet outil permet de sélectionner, de déplacer et de redimensionner les contrôles créés sur l'UserForm.

Label ou étiquette
Cet outil permet de créer une zone de texte non modifiable par l'utilisateur.

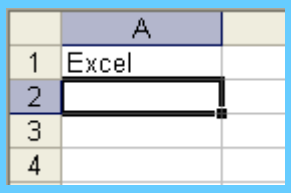


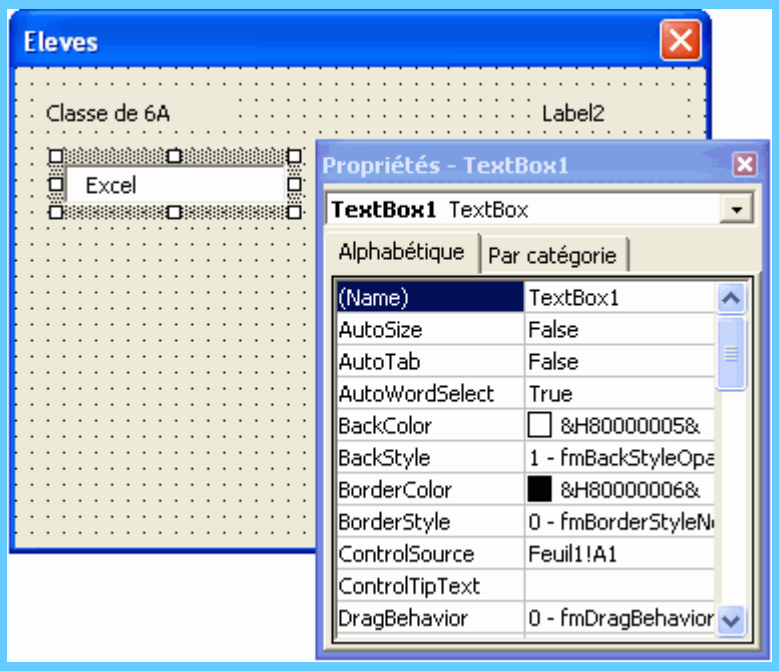
Dans cet exemple, 2 étiquettes ont été créées. Par défaut leur nom était Label1 et Label2. Pour plus de confort, elles ont été renommées Label_Classe et Label_Date. La valeur de Label_Class étant fixe, elle a été saisie dans sa propriété Caption. La valeur de Label_Date étant variable, elle peut être définie dans l'évènement Initialize de l'UserForm (renommé MaBoite).

```
Private Sub UserForm_Initialize()
    Label_Date.Caption = Date
End Sub
```

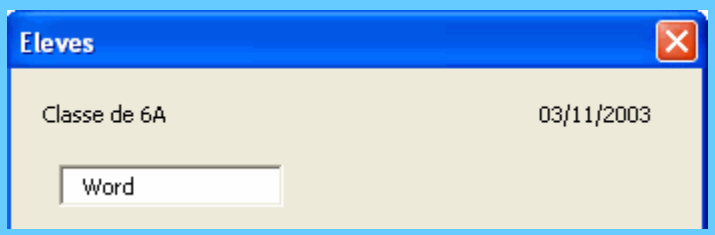
TextBox ou zone de texte

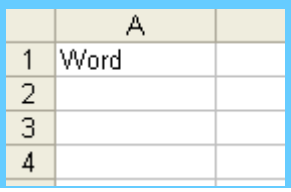
Cet outil permet de créer une zone de texte pouvant être saisie ou modifiée par l'utilisateur. Une zone de texte peut faire référence à une cellule par la propriété ControlSource.





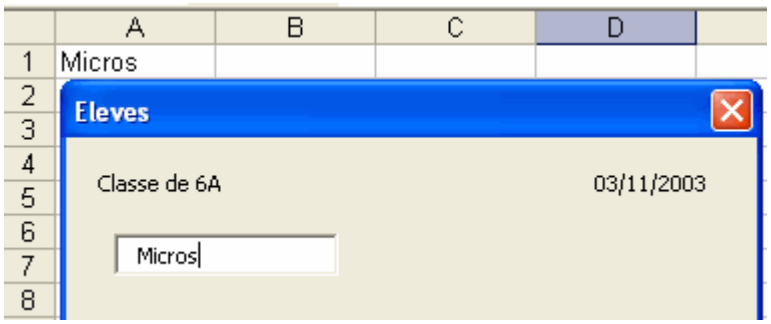
Si l'utilisateur modifie la zone de texte, la valeur de la cellule A1 sera modifiée.





La valeur de la cellule A1 peut également prendre la valeur de la zone de texte par une procédure événementielle TextBox_Change.

```
Private Sub TextBox1_Change()
    Range("A1") = TextBox1
End Sub
```

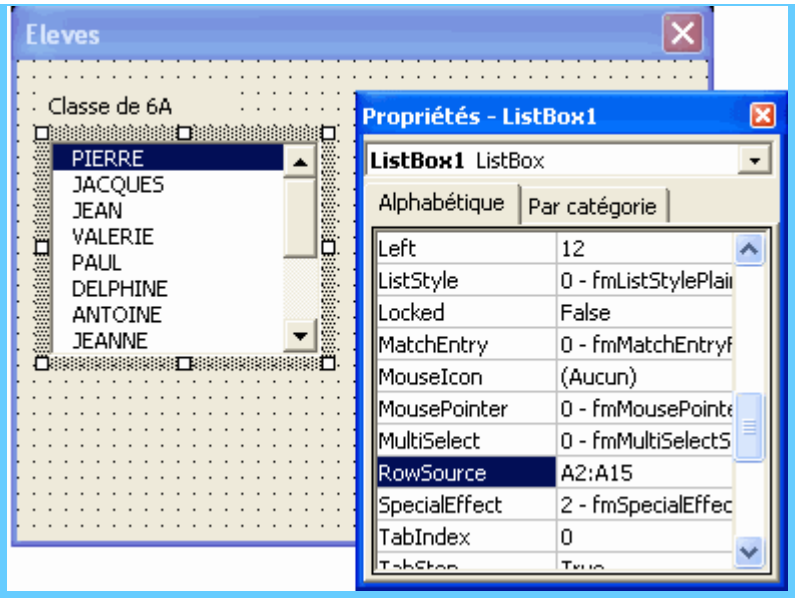


ListBox ou zone de liste

Une zone de liste permet d'afficher une liste d'éléments sélectionnables par l'utilisateur. Reprenons la liste d'élèves.

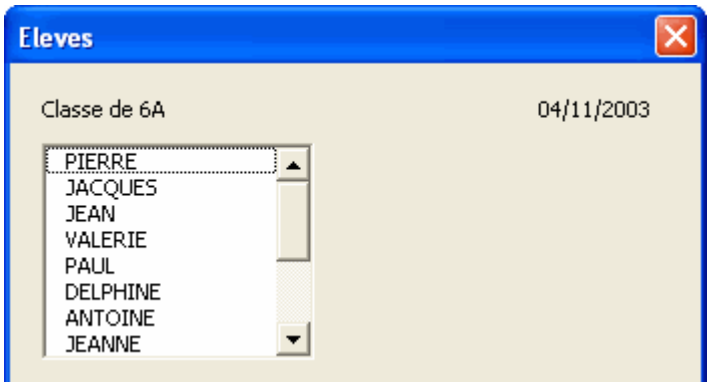
	A	B
1	ELEVE	NOTE
2	PIERRE	5
3	JACQUES	15
4	JEAN	10
5	VALERIE	12
6	PAUL	18
7	DELPHINE	13
8	ANTOINE	0
9	JEANNE	6
10	ANDRE	19
11	JOCELYNE	8
12	FELIX	12
13	JUSTIN	15
14	ETIENNE	6
15	MARIE	5

Les listes peuvent de remplir par la propriété RowSource de l'objet ListBox.

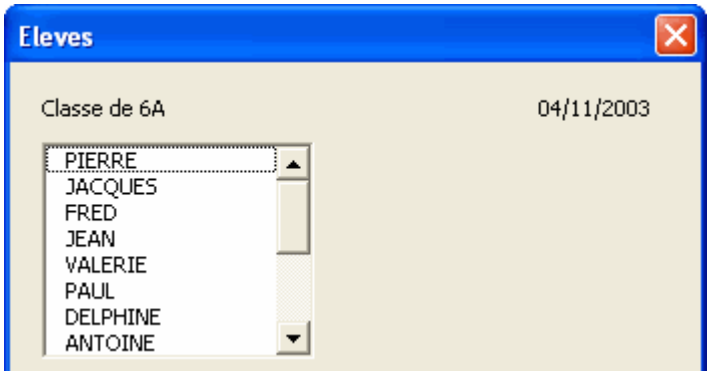


La méthode AddItem d'un objet ListBox permet d'ajouter un élément à la liste. La syntaxe est ListBox.AddItem "Texte", Index. Index correspond à l'emplacement du nouvel élément dans la liste. L'index du premier élément d'une liste a pour valeur 0. Si l'index n'est pas indiqué, le nouvel élément sera placé à la fin de la liste.

```
'La liste peut se remplir par la procédure suivante
Private Sub UserForm_Initialize()
    'Appel de la procédure Liste située dans un module
    Liste
End Sub

Sub Liste()
    Dim i As Integer
    For i = 1 To 14
        MaBoite.ListBox1.AddItem Range("A1").Offset(i)
    Next i
End Sub
```




```
'Ajout d'un élément situé en 2ème position
    MaBoite.AddItem "FRED", 2
```



La propriété ListCount d'une zone de liste renvoie le nombre d'éléments qu'elle contient, la propriété Value sa valeur et la propriété ListIndex son index.


```
Private Sub ListBox1_Change()
    Dim i As Integer, j As Integer
    Dim Val As String
    i = ListBox1.ListCount 'renvoie 15
    j = ListBox1.ListIndex  'renvoie 4
    Val = ListBox1.Value    'renvoie "VALERIE"
End Sub
```

Il est également possible de remplir une zone de liste en utilisant un tableau et la propriété List.

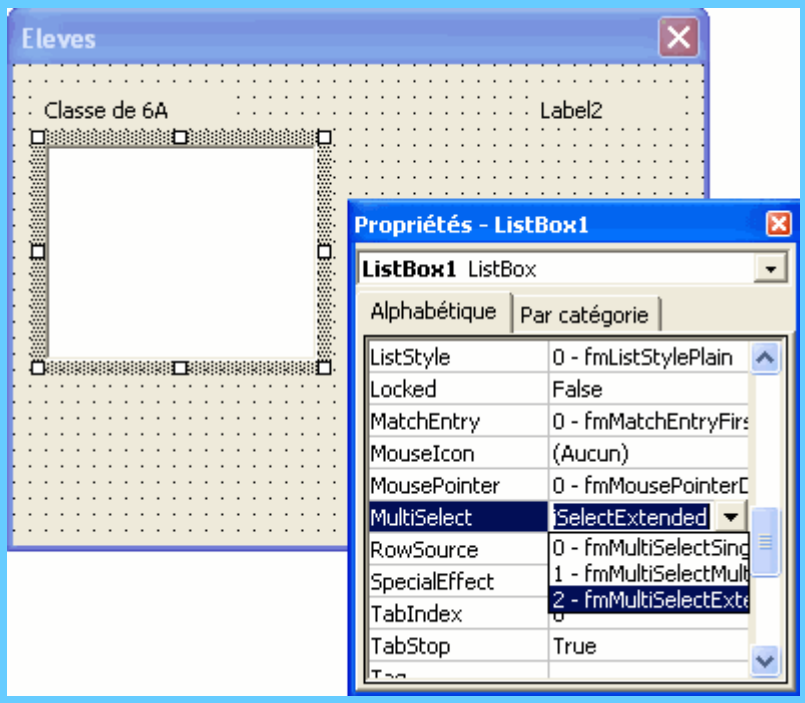
```
Sub Liste()
    Dim i As Integer
    Dim List_Eleve(1 To 14) As String
    For i = 1 To 14
        List Eleve(i) = Range("A1").Offset(i)
    Next i
    MaBoite.ListBox1.List = List_Eleve
End Sub
```

La suppression d'un élément d'une liste se fait par la méthode RemoveItem. La syntaxe est ListBox.RemoveItem Index. Index correspond à l'élément à supprimer.


```
Dim i As Integer
i = ListBox1.ListIndex  'renvoie 4
ListBox1.RemoveItem i
```

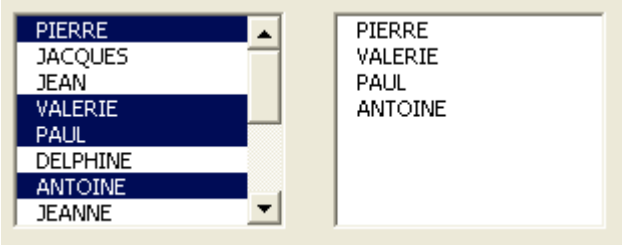
La suppression de tous les éléments d'une liste se fait par la méthode Clear. La syntaxe est ListBox.Clear.

Par défaut, l'utilisateur ne peut sélectionner qu'un seul élément de la liste. Pour permettre la sélection de plusieurs éléments, la propriété MultiSelect de la zone de texte doit être sur 1 ou sur 2.

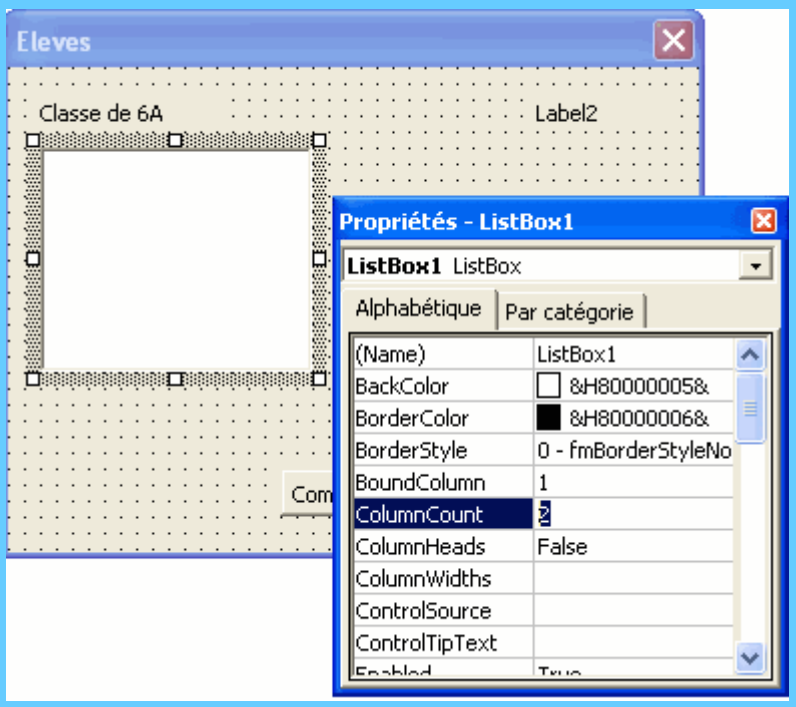


La propriété Selected(Item) détermine si un élément est sélectionné ou non. L'exemple suivant va copier les éléments sélectionnés de la ListBox1 dans la ListBox2.

```
Dim i As Integer
For i = 0 To ListBox1.ListCount - 1
    If ListBox1.Selected(i) = True Then
        ListBox2.AddItem ListBox1.List(i)
    End If
Next i
```



Une zone de liste peut contenir plusieurs colonnes. Le nombre de colonnes est défini par la propriété ColumnCount.
Dans l'exemple suivant, la zone de liste va être composée de deux colonnes.

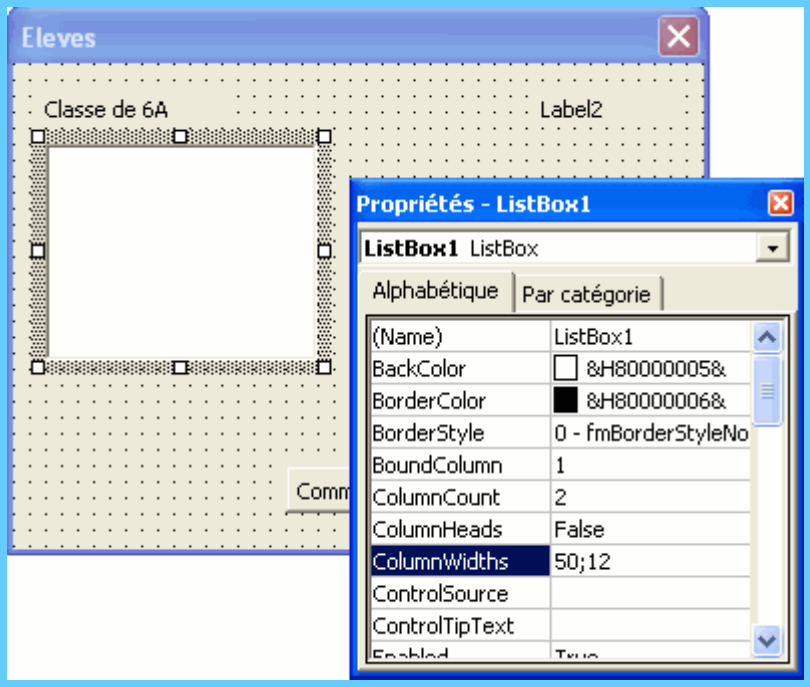


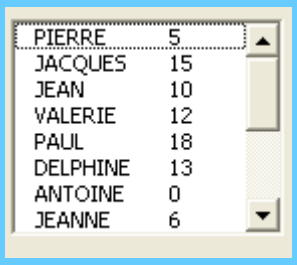
Pour remplir une zone de liste avec plusieurs colonnes, on va utiliser un tableau à plusieurs dimensions.
Dans l'exemple suivant, la zone de liste va recevoir le nom des élèves avec leurs notes.

```
Sub Liste()
    Dim i As Integer
    Dim List_Eleve(1 To 14, 1 To 2) As String
    For i = 1 To 14
        List_Eleve(i, 1) = Range("A1").Offset(i)
        List_Eleve(i, 2) = Range("B1").Offset(i)
    Next i
    MaBoite.ListBox1.List = List_Eleve
```

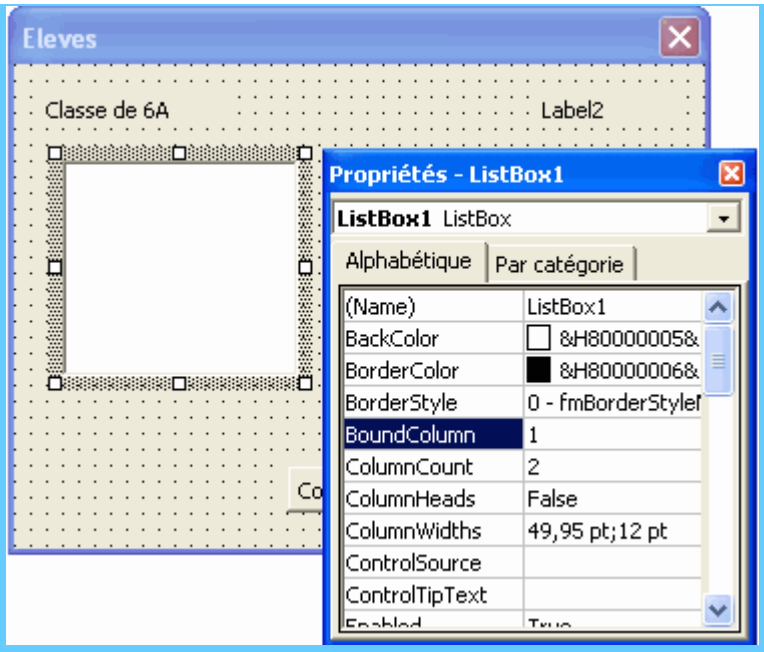
```
End Sub
```

La largeur de chaque colonne d'une zone de liste se change par la propriété ColumnWidths. Les différentes largeur sont séparées par le caractère ";".



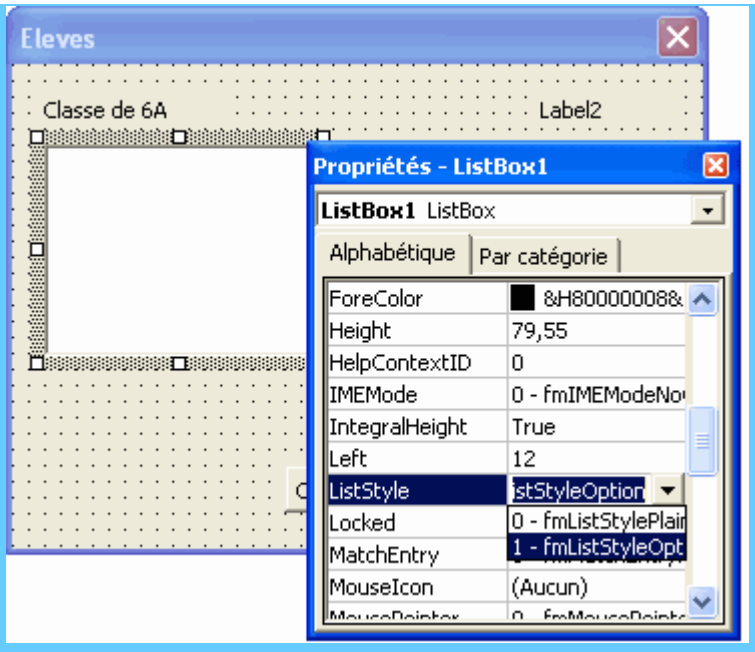


La propriété BoudColumn détermine dans quelle colonne la valeur est récupérée.

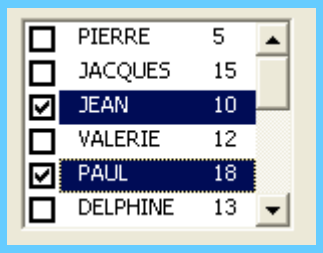



```
Dim Val As String
ListBox1.BoundColumn = 1
Val = ListBox1.Value    'renvoie "VALERIE"
ListBox1.BoundColumn = 2
Val = ListBox1.Value    'renvoie "12"
```

Il est possible de changer l'aspect d'une zone de liste par la propriété ListStyle.



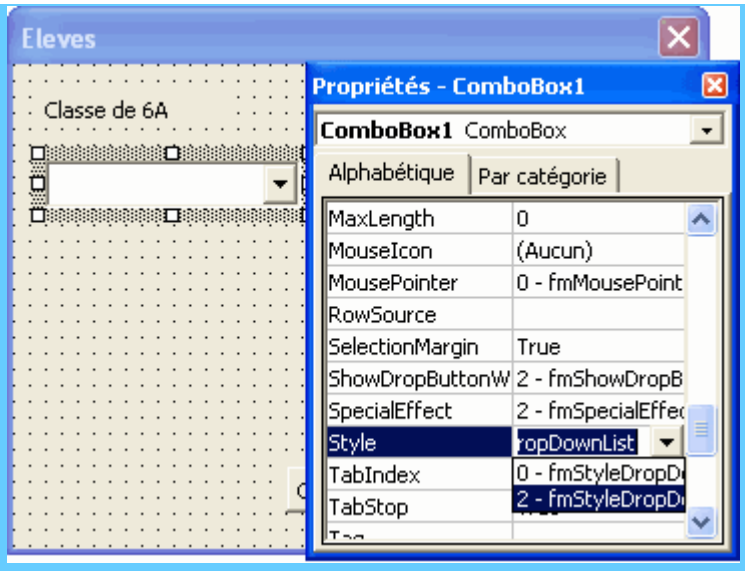
L'aspect de la zone de liste change selon la valeur de la propriété MultiSelect.



ComboBox ou liste déroulante

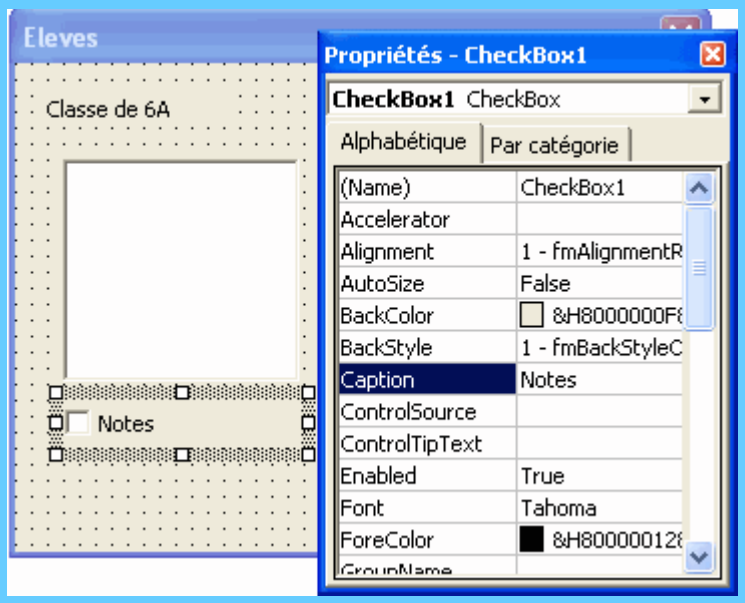
Une liste déroulante se remplit de la même façon qu'une zone de liste.
Contrairement à la zone de liste, la liste déroulante peut permettre à l'utilisateur de saisir une chaîne de caractères qui ne fait pas partie de la liste.

Si la valeur saisie ne fait pas partie de la liste ou est nulle, la propriété ListIndex de l'objet ComBox prend comme valeur -1.
Il est possible d'interdire à l'utilisateur de saisir une chaîne de caractère qui ne fait pas partie de la liste en mettent la propriété Style sur 2.



CheckBox ou case à cocher

Cet outil crée des cases que l'utilisateur peut activer ou désactiver d'un simple click

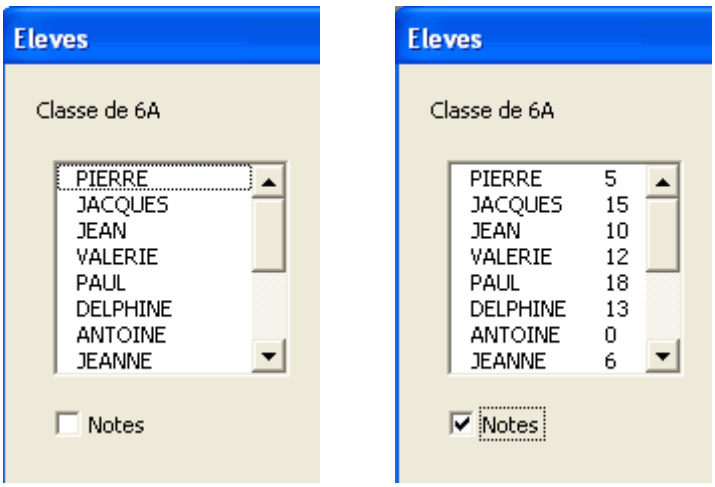


Si la case à cocher est activée, sa propriété Value prend comme valeur True, sinon elle prend comme valeur False.

Dans l'exemple suivant, si la case à cocher est activée, les notes apparaissent, sinon elles disparaissent.

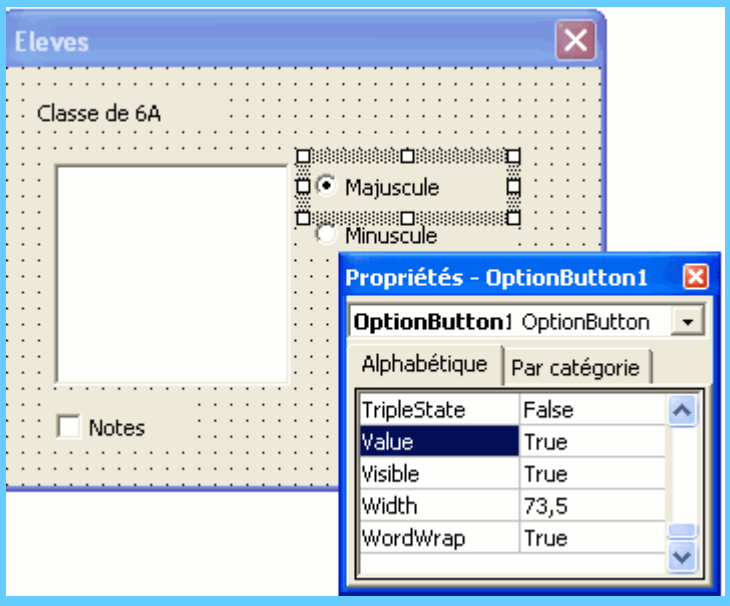
```
Private Sub UserForm_Initialize()
    Dim i As Integer
    Dim List_Eleve(1 To 14, 1 To 2) As String
    For i = 1 To 14
        List_Eleve(i, 1) = Range("A1").Offset(i)
        List_Eleve(i, 2) = Range("B1").Offset(i)
    Next i
    MaBoite.ListBox1.List = List_Eleve
End Sub
```

```
Private Sub CheckBox1_Click()
    If CheckBox1.Value = True Then
        ListBox1.ColumnCount = 2
    Else
        ListBox1.ColumnCount = 1
    End If
End Sub
```



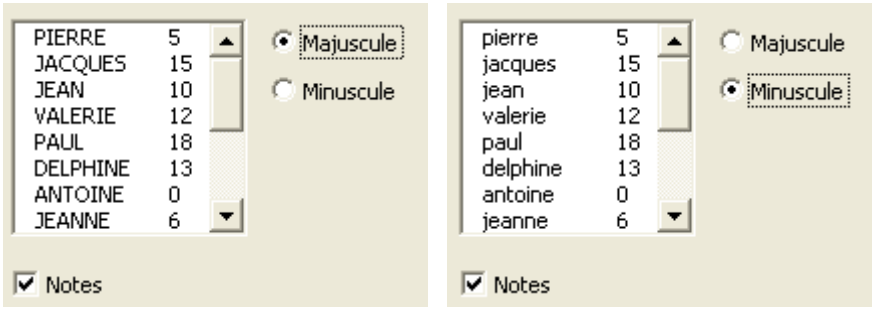

OptionButton ou bouton d'option

Cet outil crée des boutons de d'options. L'utilisateur ne peut sélectionner qu'un seul bouton d'un même groupe.



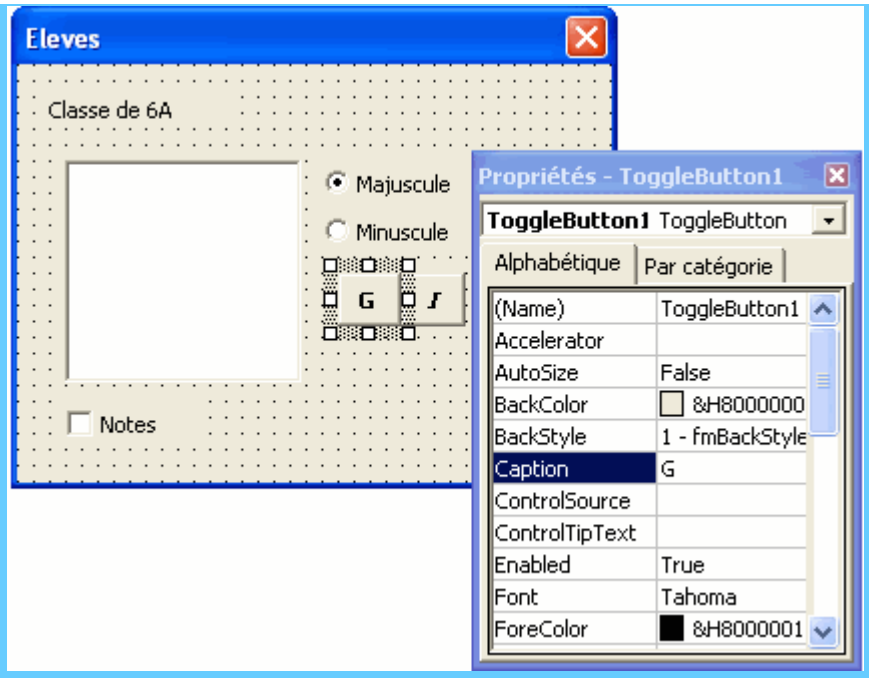
La propriété Value du bouton sélectionné prend la valeur True alors que la propriété Value des autres boutons du même groupe prend la valeur False.

```
Public Maj As Boolean

Private Sub OptionButton1_Click()
    Maj = True
    Ecrire
End Sub

Private Sub OptionButton2_Click()
    Maj = False
    Ecrire
End Sub

Sub Ecrire()
    Dim i As Integer
    For i = 0 To ListBox1.ListCount - 1
        If Maj = True Then
            'Majuscule
            ListBox1.List(i) = UCase(ListBox1.List(i))
        Else
            'Majuscule
            ListBox1.List(i) = LCase(ListBox1.List(i))
        End If
    Next i
End Sub
```



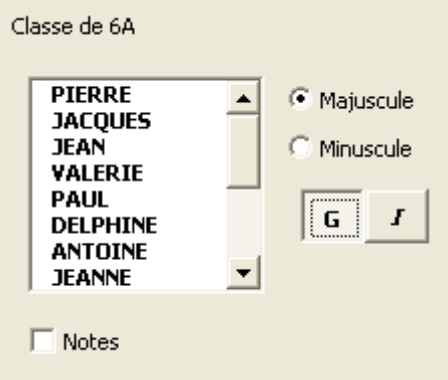
ToggleButton ou Bouton à bascule

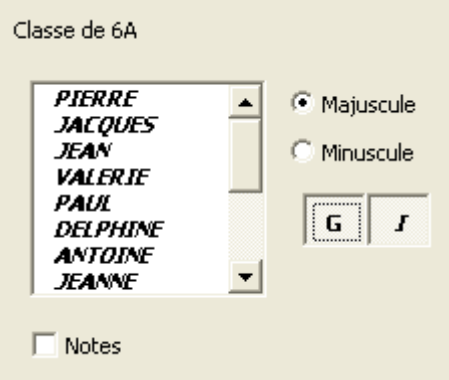
Cet outil crée un bouton qui change d'aspect à chaque click.



Si le bouton est enfoncé, sa valeur est égale à True sinon elle est égale à False.

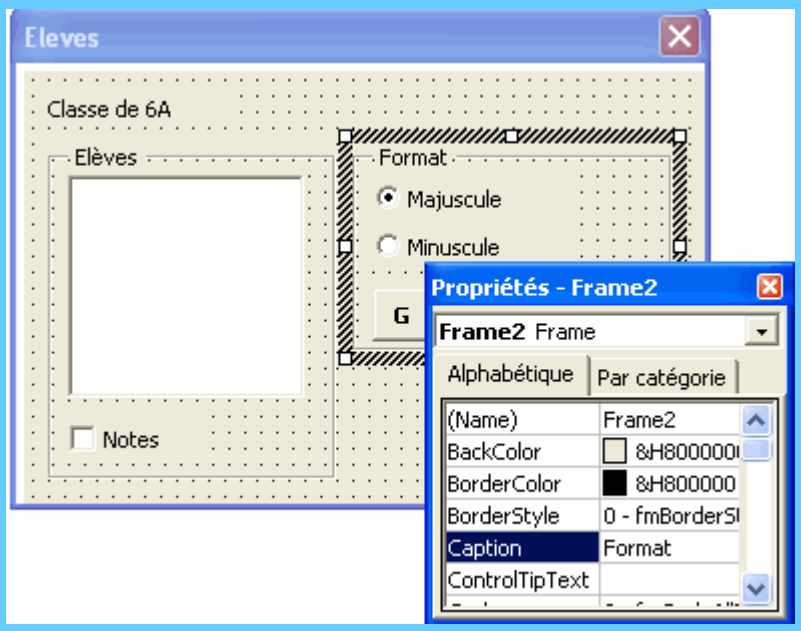
```
'Bouton Gras
Private Sub ToggleButton1_Click()
    If ToggleButton1 = True Then
        ListBox1.Font.Bold = True
    Else
        ListBox1.Font.Bold = False
    End If
End Sub
'Bouton Italic
Private Sub ToggleButton2_Click()
    If ToggleButton2 = True Then
        ListBox1.Font.Italic = True
    Else
        ListBox1.Font.Italic = False
    End If
End Sub
```



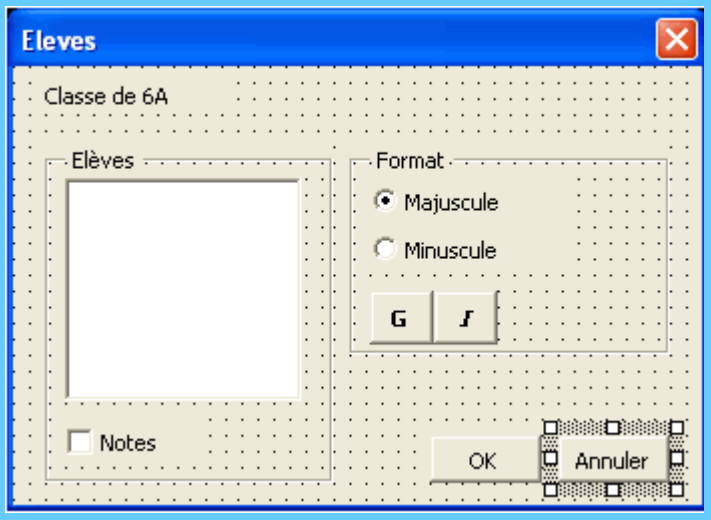


Frame ou cadre

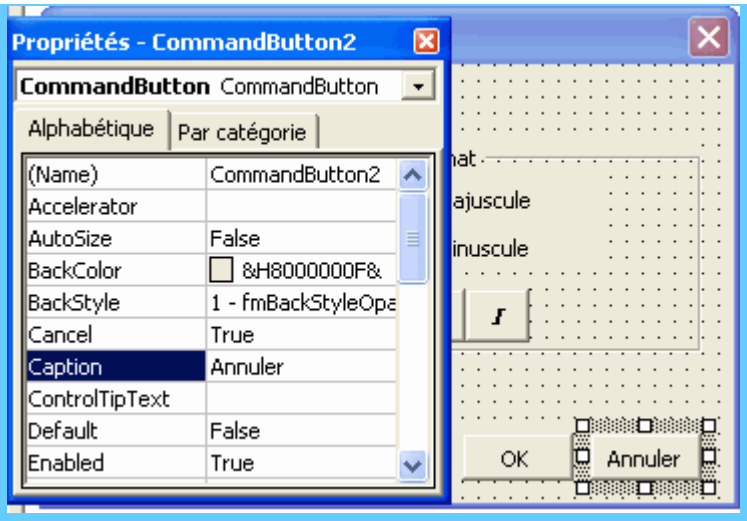
Cet outil crée des cadres permettant de grouper des contrôles.



 CommandButton ou Bouton de commande
Cet outil crée des boutons de commande tel que des boutons OK ou Annuler.

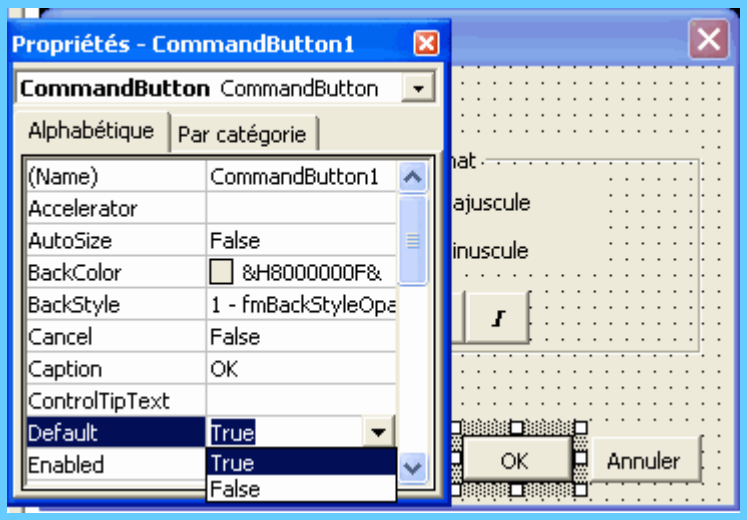


Si vous affectez la valeur True à la propriété Default d'un bouton de commande et si aucun autre contrôle n'est sélectionné, la touche Entrée équivaut à un click sur ce même bouton. De même, si vous affectez la valeur True à la propriété Cancel d'un bouton de commande, la touche Echap équivaut à un click sur le bouton.



Dans cet exemple, le fait de taper sur la touche Echap équivaut à un click sur le bouton Annuler et ferme le UserForm.

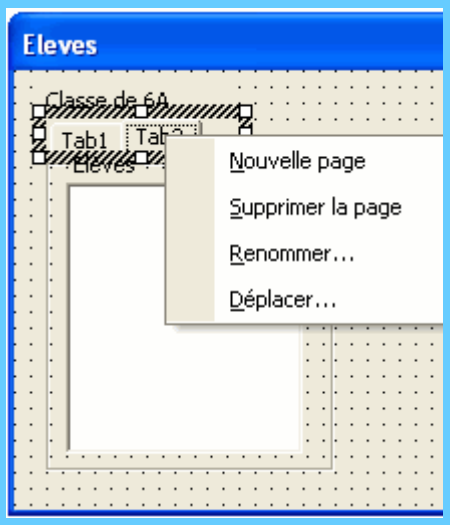
```
'Bouton Annuler
Private Sub CommandButton2_Click()
    Unload MaBoite
End Sub
```



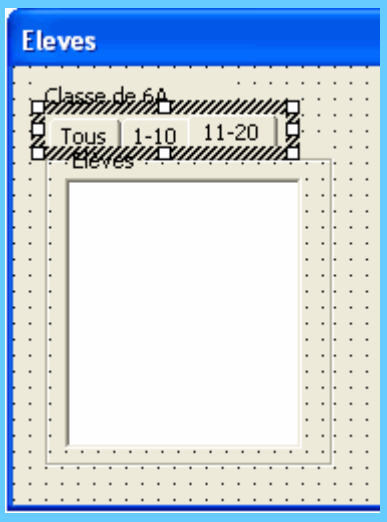
Dans cet exemple, le fait de taper sur la touche Entrée équivaut à un click sur le bouton OK si aucun autre contrôle n'est sélectionné et met à jour la liste dans la feuille de calcul.

```
'Bouton OK
Private Sub CommandButton1_Click()
    MAJListe
End Sub

Sub MAJListe()
    Dim NreENreg as Integer
    NbreEnreg = ListBox1.ListCount
    Range("A2:B" & NbreEnreg + 1) = ListBox1.List
End Sub
```

TabStrip ou étiquette d'onglet
Cet outil crée des étiquettes d'onglet pour des pages identiques.
Par défaut, le nombre d'onglets d'un nouveau TabStrip est de 2 et sont nommés Tab1 et Tab2. Un simple click avec le bouton droit de la souris permet d'en ajouter, de les renommer, de les déplacer ou de les supprimer.



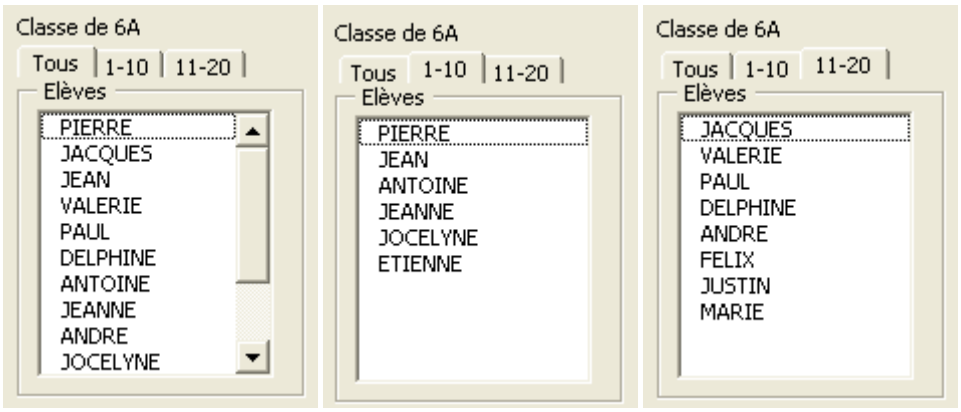
Dans l'exemple suivant, ajoutons un TabStrip avec 3 onglets permettant de classer les élèves suivant leurs notes.



L'onglet sur lequel clique l'utilisateur est déterminé par la propriété Value du Tab Strip qui prend comme valeur 0 si l'utilisateur clique sur le premier onglet, 1 si il clique sur le second, 3 si il clique sur le troisième ...

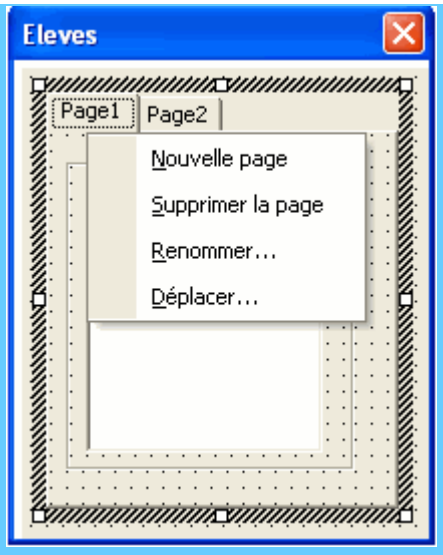
```
Private Sub TabStrip1_Click()
    OptNote = TabStrip1.Value
```

```
    Liste
End Sub

Public OptNote As Integer

Sub Liste()
    Dim Note As Integer
    Dim Eleve As String
    Dim i As Integer
    Dim NoteMini As Integer
    Dim NoteMaxi As Integer
    MaBoite.ListBox1.Clear 'Efface le contenu de la liste
    Select Case OptNote
    Case 0 'Toutes les notes(onglet 1)
        NoteMini = 0
        NoteMaxi = 20
    Case 1 'Notes de 0 à 10(onglet2)
        NoteMini = 0
        NoteMaxi = 10
    Case 2 'Notes de 11 à 20(onglet3)
        NoteMini = 11
        NoteMaxi = 20
    End Select
    For i = 1 To 14
        Note = Range("B1").Offset(i)
        If Note >= NoteMini And Note <= NoteMaxi Then
            Eleve = Range("A1").Offset(i)
            MaBoite.ListBox1.AddItem Eleve
        End If
    Next i
End Sub
```

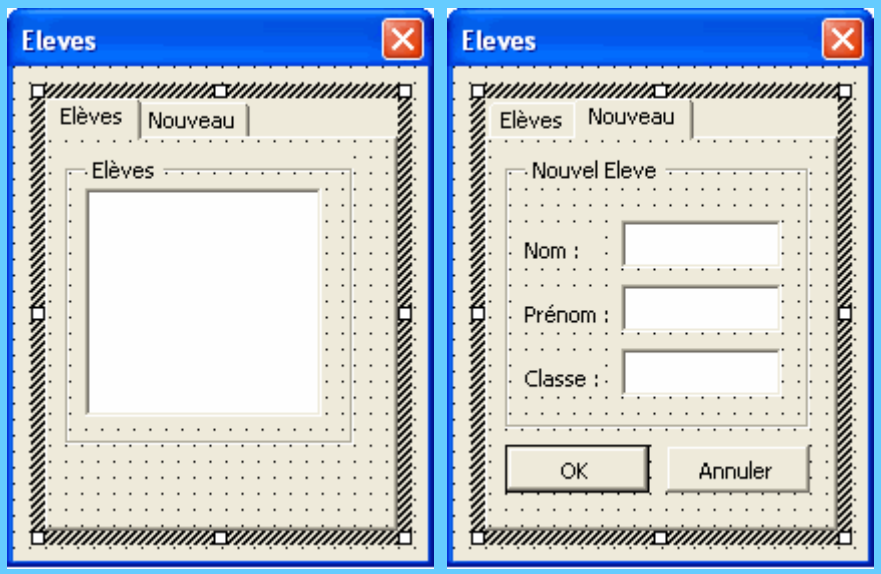


 MultiPage

Un multipage peut être comparé à plusieurs UserForm dans le même. Tout comme le TabStrip, le multipage contient des onglets mais à chaque onglet correspond une nouvelle page qui contient des contrôles différents. Sa création est identique à la création d'un TabStrip.

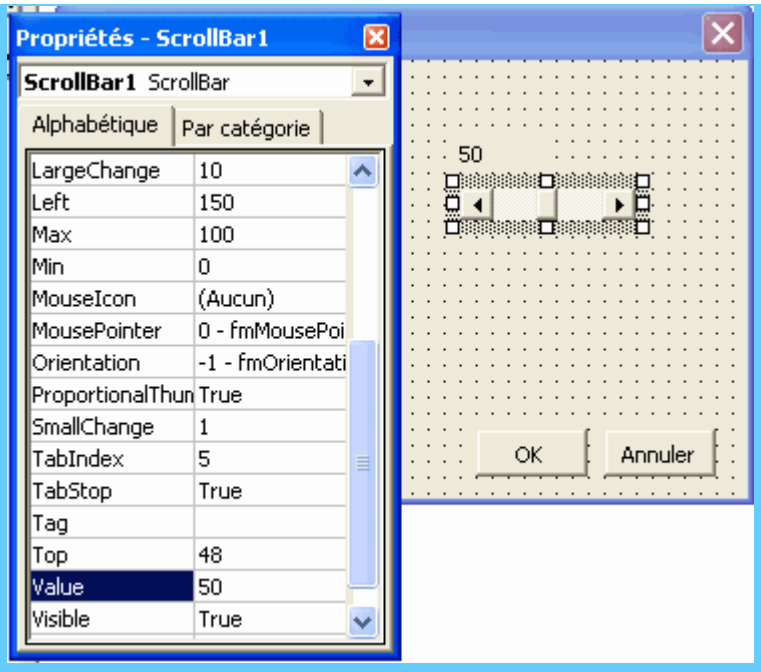


L'onglet sur lequel clique l'utilisateur est déterminé par la propriété Index du multipage qui prend comme valeur 0 si l'utilisateur clique sur le premier onglet, 1 si il clique sur le second, 3 si il clique sur le troisième ...



 ScrollBar ou Barre de défilement

Une barre de défilement peut être horizontale ou vertical selon son redimensionnement. L'exemple suivant se compose d'une barre de défilement et d'une étiquette qui reçoit sa valeur.

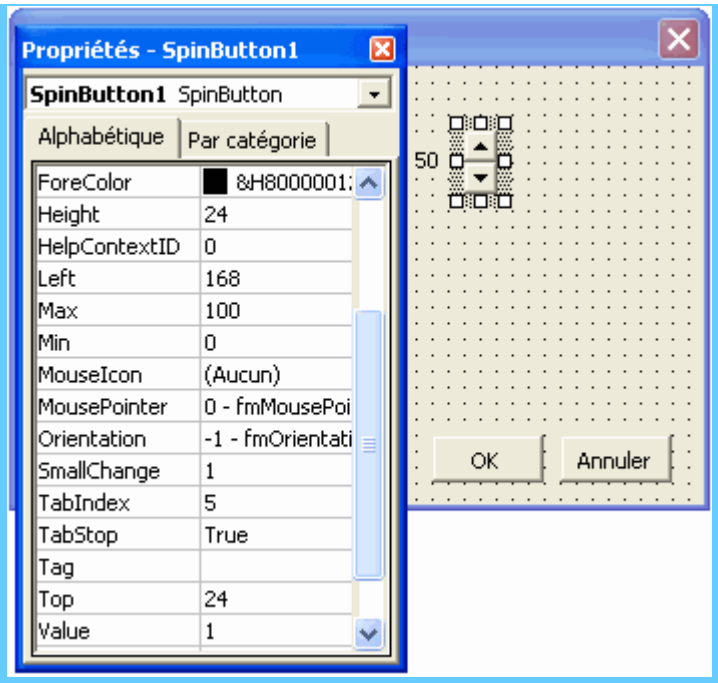


La valeur mini d'une barre de défilement se définie par sa propriété Min, sa valeur maxi par sa propriété Max et sa valeur par sa propriété Value.
La propriété LargeChange définie le changement de valeur lorsque l'utilisateur clique entre le curseur et l'une des flèches.
La propriété SmallChange définie le changement de valeur lorsque l'utilisateur clique sur l'une des deux flèches.
La propriété Delay définie le temps (en millisecondes) entre chaque changement lorsque l'utilisateur reste appuyer sur le Scrollbar.
L'étiquette prend la valeur de la barre de défilement par la procédure suivante:

```
Private Sub ScrollBar1_Click()
    Label1 = ScrollBar1.Value
End Sub
```

 SpinButton ou Bouton rotatif

Le bouton rotatif possède presque les mêmes propriétés qu'une barre de défilement. Il ne peut cependant incrémenter ou décrémenter un nombre que de la même valeur (définie dans sa propriété Value) à chaque fois.

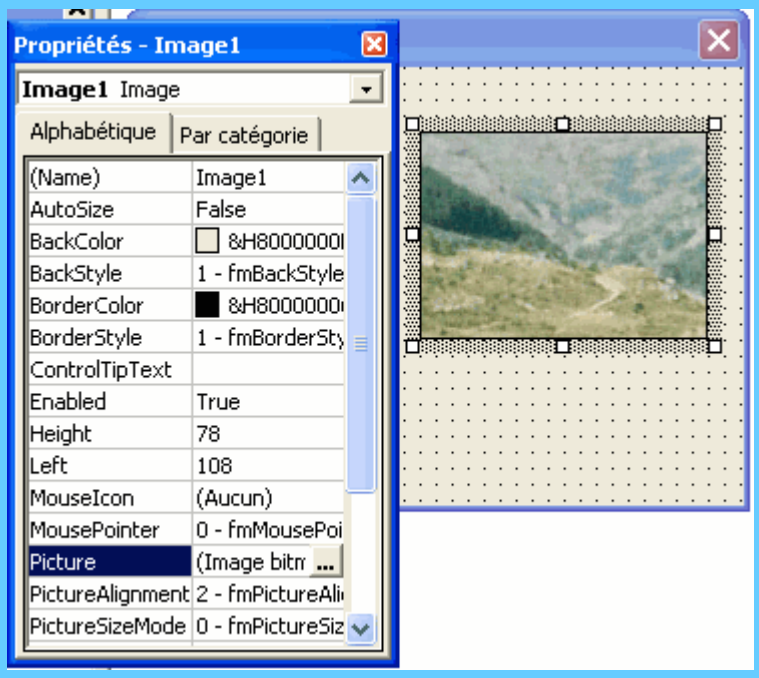



```
Private Sub SpinButton1_Click()
    Label1 = SpinButton1.Value
End Sub
```

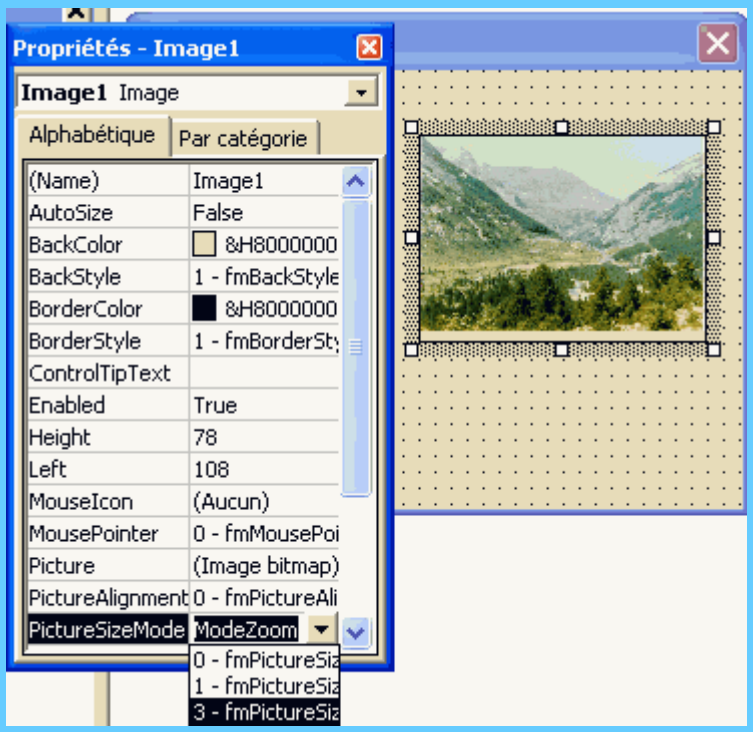
Image

Cet outil permet d'ajouter une image sur un UserForm.

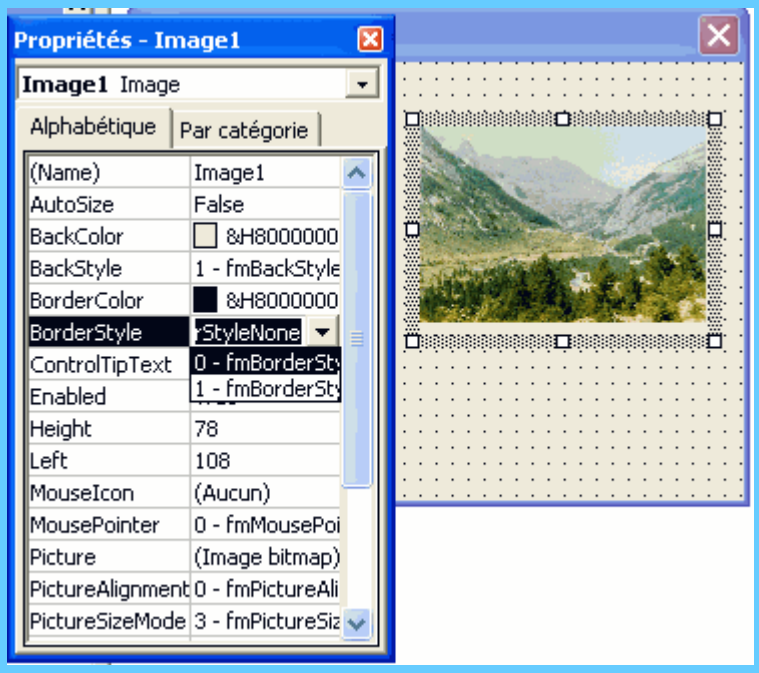
La sélection de l'image à placer se fait en cliquant sur ... de la propriété Picture.



La propriété PictureSizeMode permet de redimensionnner l'image.



La propriété BorderStyle permet de supprimer le cadre autour de l'image.



Le code VB permet également de charger ou de décharger une image par la propriété Picture.

```
Dim Photo As String
Image1.Picture = LoadPicture() 'Décharge l'image
Photo = "c:\cheminphoto\photo.jpg"
```

```
Image1.Picture = LoadPicture(Photo) 'charge l'image
```

Le contrôle Image supporte les formats d'image bmp, cur, gif, ico, jpg, et wmf.

Exemples - Fichiers-

Nombre de fichiers d'un répertoire dans la cellule A1

```
Sub Macro()
    'Appel de la fonction
    Range("A1") = Nbre_Fich("c:\Excel\", "*.xls")
End Sub

Function Nbre_Fich(Repert As String, Optional Term As String) As Integer
    Dim Fichier As String
    'Si il existe, la fonction Dir renvoie le nom du 1er fichier.
    Fichier = Dir(Repert & Term)
    Do While Fichier <> ""
        Nbre_Fich = Nbre_Fich + 1
        'Rappel de la fonction Dir sans arguments pour faire appel aux
        'fichiers suivants.
        Fichier = Dir
    Loop
End Function
```

HAUT FERMER

Lister les fichiers dont la date d'enregistrement est > à une date

```
Sub Fichiers_Dates()
    Dim Repert As String
    Dim i As Integer
    Dim Date1 As Date, Date2 As Date
    Dim Fichier As String
    Repert = "c:\excel\"
    Fichier = Dir(Repert)
    'Date de référence
    Date1 = Range("A1")
    Do While Fichier <> ""
        'Date du dernier enregistrement du fichier
        Date2 = FileDateTime(Repert & Fichier)
        If Date2 > Date1 Then
            Range("B1").Offset(i) = Fichier
            Range("C1").Offset(i) = Date2
            i = i + 1
        End If
        Fichier = Dir
    Loop
End Sub
```

	A	B	C
1	22/06/2003	Classeur2.xls	22/08/2003 15:16
2		classeur3.xls	05/11/2003 14:50
3		classeur4.xls	23/06/2003 18:26
4			

HAUT FERMER

Copier des fichiers d'un dossier à un autre

```
'Cette procédure copie tous les fichiers du dossier "c:\excel\" dans le
'dossier "c:\excel2" par l'instruction FileCopy.
Sub Copie_Fichiers()
    Dim Repert1 As String, Repert2 As String
    Dim Fichier As String
    Repert1 = "c:\excel\"
    Repert2 = "c:\excel2\"
    Fichier = Dir(Repert1)
    Do While Fichier <> ""
        FileCopy Repert1 & Fichier, Repert2 & Fichier
        Fichier = Dir
    Loop
End Sub
```

HAUT FERMER

Supprimer des fichiers

```
'Cette procédure supprime tous les fichiers du dossier "c:\excel\" par
'l'instruction Kill.
Sub Supp_Fichiers()
    Dim Repert As String
    Dim Fichier As String
    Repert = "c:\excel\"
    Fichier = Dir(Repert)
    Do While Fichier <> ""
        Kill Repert & Fichier
        Fichier = Dir
    Loop
End Sub
```

HAUT FERMER

Renommer des fichiers

```
'Cette procédure renomme le fichier "classeur1.xls" du dossier
'"c:\excel\" en "dossier1.xls" par 'l'instruction Name.
Sub Renom_Fichiers()
    Dim Repert As String
    Dim Fichier1 As String, Fichier2 As String
    Repert = "c:\excel\"
    Fichier1 = "classeur1.xls"
    Fichier2 = "dossier1.xls"
    Name Repert & Fichier1 As Repert & Fichier2
End Sub
```

HAUT FERMER

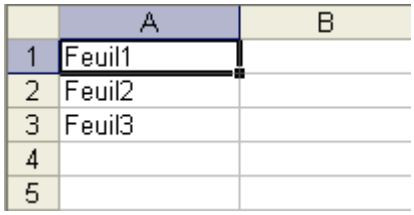
Empêcher l'enregistrement d'un fichier

```
'Cette procédure évènementielle interdit l'enregistrement du fichier
Private Sub Workbook_BeforeSave(ByVal SaveAsUI As Boolean, Cancel As
Boolean)
    MsgBox "Impossible d'enregistrer"
    Cancel = True
End Sub
```

HAUT FERMER

Afficher le nom des feuilles d'un classeur

```
'Cette procédure affiche le nom des feuilles d'un classeur
Sub Aff_Feuilles()
    Dim Feuille As Worksheet
    Dim i As Integer
    For Each Feuille In Worksheets
        Range("A1").Offset(i) = Feuille.Name
        i = i + 1
    Next
End Sub
```



	A	B
1	Feuil1	
2	Feuil2	
3	Feuil3	
4		
5		

HAUT FERMER

Enregistrer un classeur sous le nom de la valeur d'une cellule

```
'Cette procédure enregistre le classeur actif sous la valeur de la
'cellule "A1" qui contient "janvier.xls".
Sub Enreg_Fichier()
    Dim NomFichier As String
    NomFichier = Range("A1")
    ActiveWorkbook.SaveAs "c:\excel\" & NomFichier
End Sub
```

HAUT FERMER

Test l'existence d'un fichier

```
'Cette procédure teste si le fichier "c:\Excel\Classeur1.xls" existe
Sub Test_Fichier()
    Dim Fichier As String
    Fichier = Dir("c:\Excel\Classeur1.xls")
    If Fichier <> "" Then
        'le fichier existe
    Else
        'le fichier n'existe pas
    End If
End Sub
```

HAUT FERMER

Fermer tous les classeurs ouverts

```
'Cette procédure ferme tous les fichiers sauf le fichier "ferm.xls"
Sub Ferm_Fichiers()
    Dim i As Integer, j As Integer
    Dim TabFichier() As String
    j = Workbooks.Count 'Nombre de fichiers
    ReDim TabFichier(j)
    For i = 1 To j
        'nom des fichiers dans le tableau
        TabFichier(i) = Workbooks(i).Name
    Next i
    'fermeture des fichiers
    For i = 1 To j
        If TabFichier(i) <> "ferm.xls" Then
            Workbooks(TabFichier(i)).Close
        End If
    Next i
End Sub
```

HAUT FERMER

Mettre en majuscule tous les onglets d'un classeur

```
Sub Maj_Onglets()
    Dim i As Integer
    For i = 1 To Sheets.Count
        Sheets(i).Name = UCase(Sheets(i).Name)
    Next i
End Sub
```

HAUT FERMER

Répertoire + nom du dossier dans le pied de page

```
'Cette procédure met le nom du répertoire dans le pied de page à gauche
'et le nom du fichier à droite
Sub Pied_Page()
    Dim Repert As String
    Dim Fichier As String
```

```
    Repert = ActiveWorkbook.Path
    Fichier = ActiveWorkbook.Name
    With ActiveSheet.PageSetup
        .LeftFooter = Repert
        .RightFooter = Fichier
    End With
End Sub
```

HAUT FERMER

Réduire tous les classeurs

```
Sub Red_Class()
    Dim i As Integer
    For i = 1 To Workbooks.Count
        Workbooks(i).Activate
        ActiveWindow.WindowState = xlMinimized
    Next i
End Sub
```

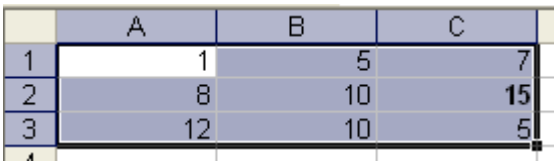
HAUT FERMER

Exemples - Cellule-

Valeur maximale d'une sélection

```
'Cette procédure recherche puis met en gras la cellule qui possède la
'plus grande valeur
Sub ValMaxi()
    Dim Cel As Range
    Dim Val As Integer
    Dim Adr As String
    For Each Cel In Selection
        If Val < Cel Then
```

```
            Val = Cel 'Valeur de la cellule
            Adr = Cel.Address 'Adresse de la cellule
        End If
    Next
    Range(Adr).Font.Bold = True
    'La plus grande valeur est contenu dans la variable Val
End Sub
```

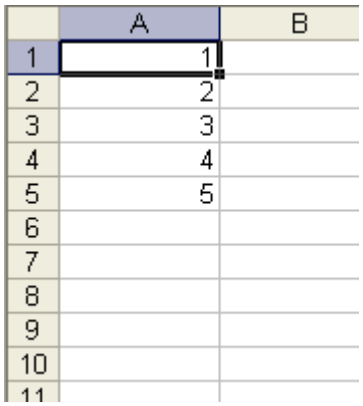


	A	B	C
1	1	5	7
2	8	10	**15**
3	12	10	5

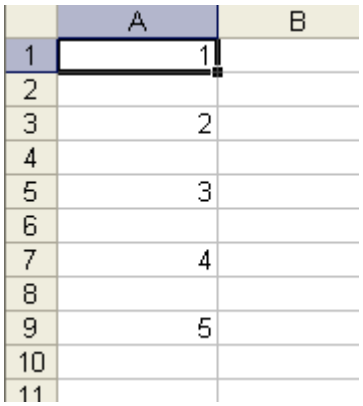
HAUT FERMER

Insérer une ligne entre chaque cellule

```
Sub ValMaxi()
    Dim i As Integer
    i = 1
    Do While Range("A1").Offset(i) <> ""
        Rows(i + 1).Insert
        i = i + 2
    Loop
End Sub
```



	A	B
1	1	
2	2	
3	3	
4	4	
5	5	
6		
7		
8		
9		
10		



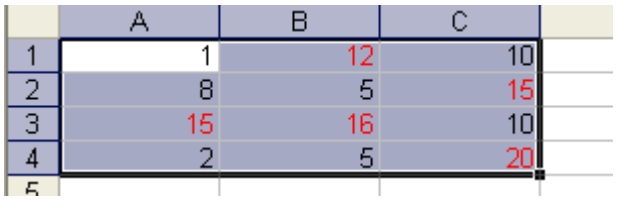
	A	B
1	1	
2		
3	2	
4		
5	3	
6		
7	4	
8		
9	5	
10		

HAUT FERMER

Mettre de couleur rouge les cellules d'une sélection dont le nombre est > 10

```
Sub Cel_Rouge()
    Dim Cel As Range
    Dim Rouge As Byte, Vert As Byte, Bleu As Byte
```

```
    Rouge = 255
    Vert = 0
    Bleu = 0
    For Each Cel In Selection
        If Cel > 10 Then
            Cel.Font.Color = RGB(Rouge, Vert, Bleu)
        End If
    Next
End Sub
```

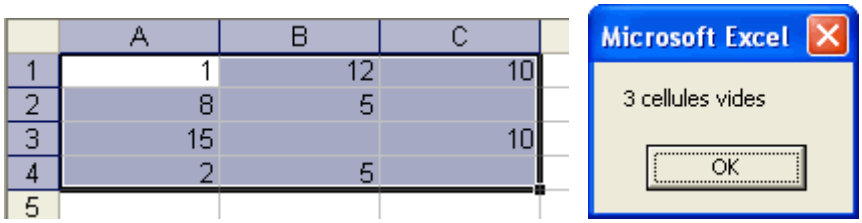


	A	B	C
1	1	12	10
2	8	5	15
3	15	16	10
4	2	5	20

HAUT FERMER

Nombre de cellules vides d'une sélection

```
Sub Cel_Vide()
    Dim Cel As Range
    Dim i As Integer
    For Each Cel In Selection
        If Cel = "" Then
            i = i + 1
        End If
    Next
    MsgBox i & " cellules vides"
End Sub
```

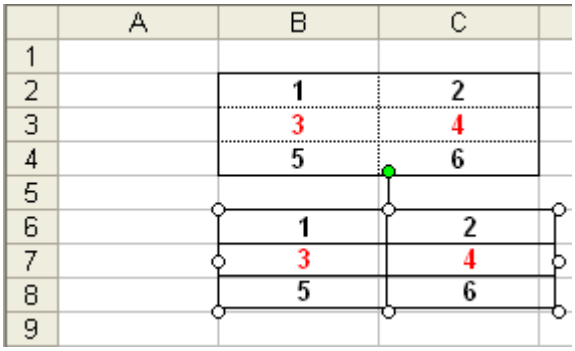


HAUT FERMER

Copier une sélection en image

```
Sub Copy_Sel_Image()
    Range("B2:C4").Copy
    Range("B6").Select
    ActiveSheet.Pictures.Paste
```

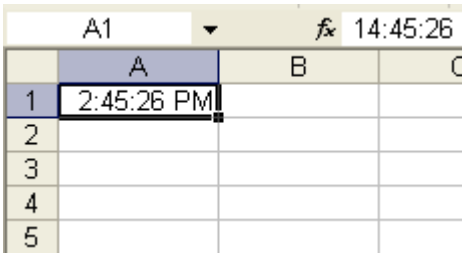
```
    Application.CutCopyMode = False
End Sub
```



HAUT FERMER

Heure dans une cellule

```
'Cette procédure va créer une horloge dans la cellule A1.
Sub Heure()
    Application.OnTime Now + TimeValue("00:00:01"), "Heure"
    Range("A1") = Time
End Sub
```




```
'Cette procédure arrête l'horloge
Sub Arret()
    Application.OnTime Now + TimeValue("00:00:01"), "Heure", , False
End Sub
```

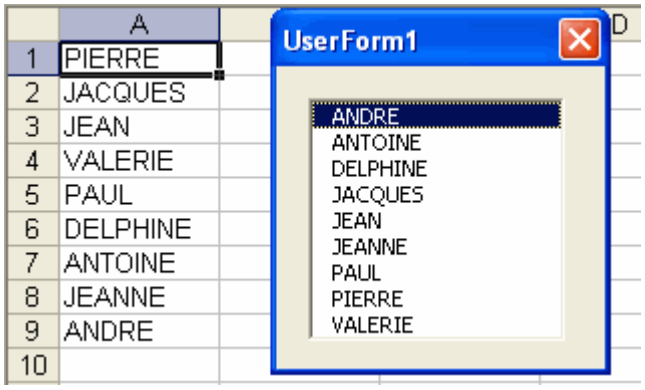
HAUT FERMER

Site Classé

Exemples - UserForms -

Mettre en ordre alphabétique une listbox

```
Sub List_Alphab()
    Dim i As Integer, j As Integer
    Dim Entree As String
    Dim Cel As Range
    Set Cel = Range("A1")
    'Pour chaque enregistrement
    For i = 0 To Cel.End(xlDown).Row - 1
        'Récupère la valeur
        Entree = Cel.Offset(i)
        With UserForm1
            'Pour chaque valeur de la listBox
            For j = 0 To .ListBox1.ListCount - 1
                'Si la valeur de la listbox est > à la valeur à entrer
                'on récupère l'index j et on sort de la boucle
                If .ListBox1.List(j) > Entree Then
                    Exit For
                End If
            Next j
            'ajout de la valeur à son emplacement spécifié par l'index j
            .ListBox1.AddItem Entree, j
        End With
    Next i
    UserForm1.Show
End Sub
```



HAUT FERMER

Fermer le classeur lors de la fermeture d'un Userform

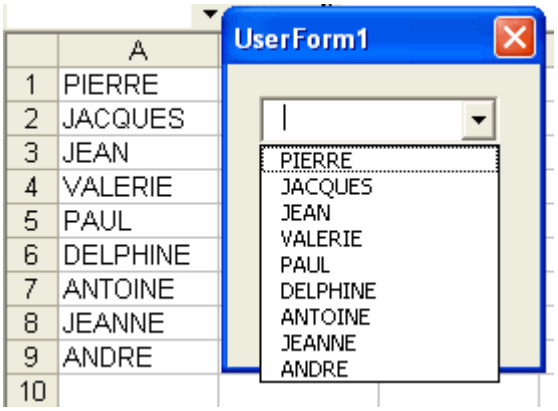
```
'Cette procédure évènementielle ferme le classeur sans l'enregistrer
lors de la fermeture de l'UserForm
```

```
Private Sub UserForm_QueryClose(Cancel As Integer, CloseMode As Integer)
    Workbooks("nomduclasseur.xls").Close False
End Sub
```

HAUT FERMER

Dimensionner une zone de liste suivant le nombre d'enreg

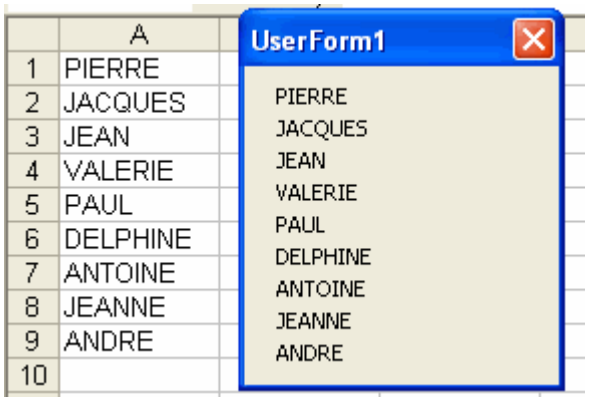
```
Sub Nbre_ZoneListe()
    Dim NbreEnreg As Integer
    With UserForm1
        .ComboBox1.RowSource = "A1:A9"
        NbreEnreg = .ComboBox1.ListCount
        .ComboBox1.ListRows = NbreEnreg
        .Show
    End With
End Sub
```



HAUT FERMER

Ecrire dans plusieurs contrôles en utilisant une boucle

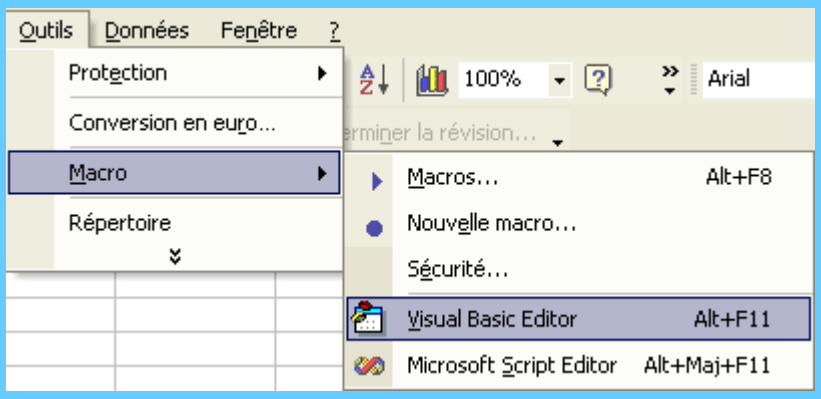
```
Sub Ecrir_Control()
    Dim i As Integer
    Dim Cel As Range
    Set Cel = Range("A1")
    With UserForm1
        For i = 0 To Cel.End(xlDown).Row - 1
            .Controls("Label" & i + 1) = Cel.Offset(i)
        Next i
        .Show
    End With
End Sub
```



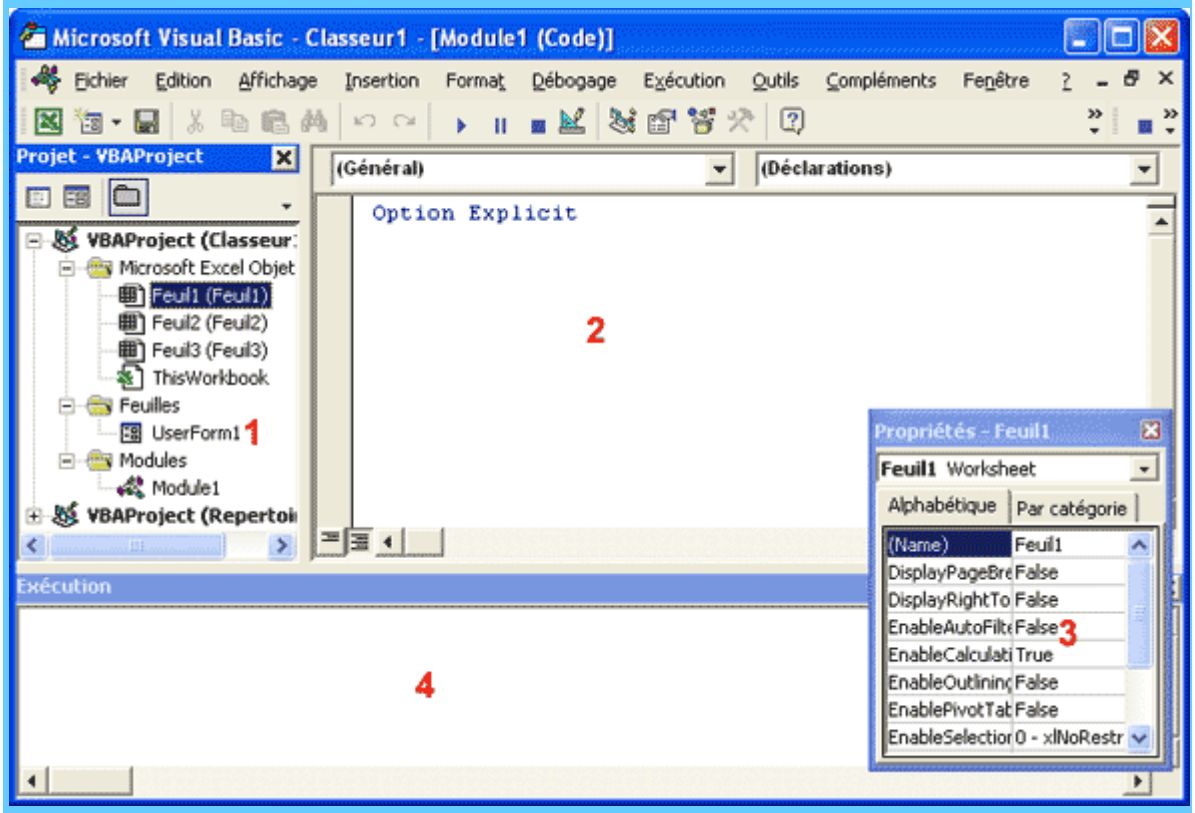
Cours VBA - L'éditeur de macros -

Ouverture de l'éditeur de macros :

L'éditeur de macro, ou VBE (Visual Basic Editor) est l'environnement de programmation de VBA. Il se lance par le menu "Outils-Macro-Visual-Basic-Editor- ou par le raccourci clavier "Alt+F11".



Les principales fenêtres de VBE :



1 - Fenêtre VBAProject. Elle présente les différents projets ouverts et permet de naviguer facilement entre vos différentes feuilles de codes VBA.
2 - Fenêtre Code. C'est l'endroit ou vous allez saisir votre code VBA.
3 - Fenêtre Propriétés. Propriétés de l'objet sélectionné.
4 - Fenêtre Exécution. Elle permet de tester une partie du code. Elle peut s'avérer très utile pour voir comment s'exécutent certaines lignes de code.

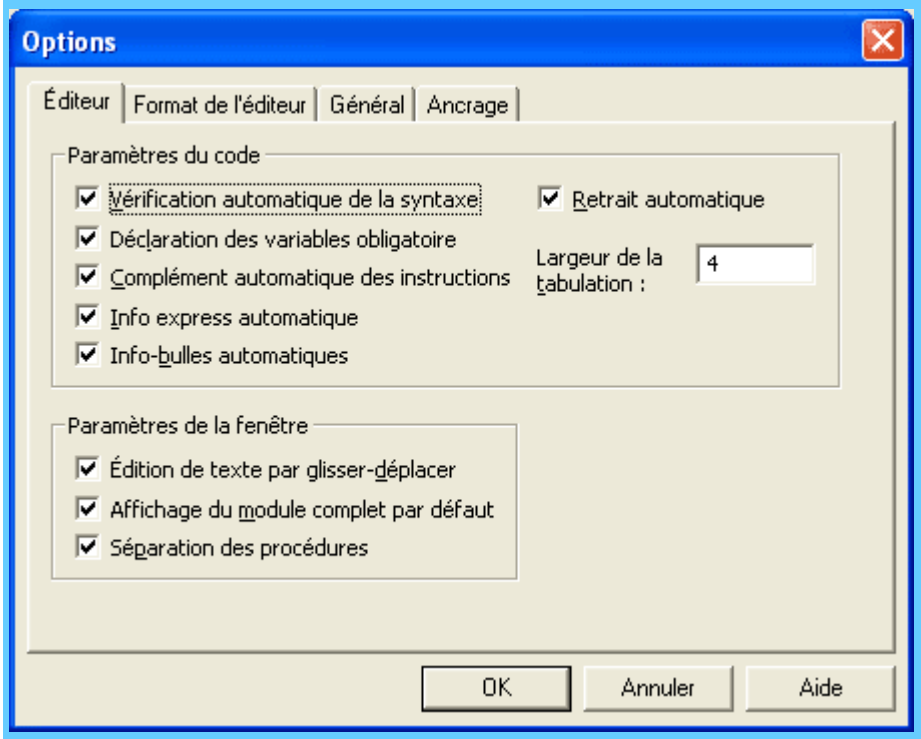
Il est fort probable que l'aspect de votre éditeur de macros soit différent. Il est en effet personnalisable car chaque fenêtre peut être masquée puis réaffichée par le menu "Affichage". Cependant, cette configuration vous permettre de débuter de façon confortable l'écriture de vos premières macros.

Bien configurer l'éditeur de macros :

Il est important de bien configurer l'éditeur de macros. En effet, VBE peut vous aider dans l'écriture de votre code et le mettre en forme de façon à ce qu'il soit plus facile à lire.

Sous VBE, lancer le menu "Outils-Options" :

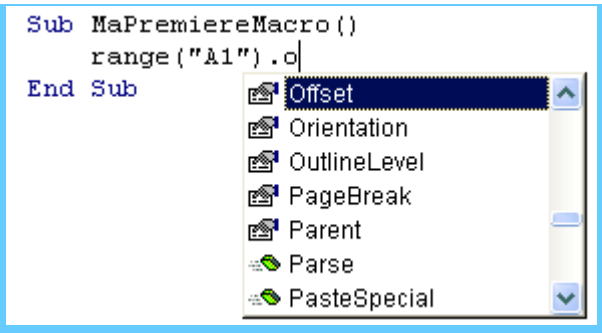
1 - Onglet Editeur :



Vérification automatique de la syntaxe :
Vérification automatiquement de la syntaxe lors de la saisie d' une ligne de code.

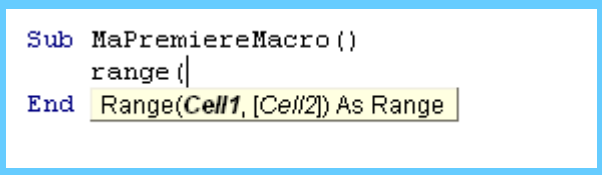
Déclarations de variables obligatoires :
Sous VB, la déclaration de variables n'est pas obligatoire. Cependant, je vous conseille de cocher cette option. De plus amples informations au sujet des variables seront disponibles dans le cours "Les variable". Si la case est cochée, l'instruction "Option Explicit" est ajoutée dans les déclarations générales de tout nouveau module.

Complément automatique des instructions :
Cette option permet à VBE de vous aider dans la saisie de votre code :



Vous comprendrez très vite son utilité lorsque vous saisirez vos premières lignes de codes.

Info express automatique :
Encore une option très utile. Elle affiche les différents arguments que possède la fonction que vous venez de taper :

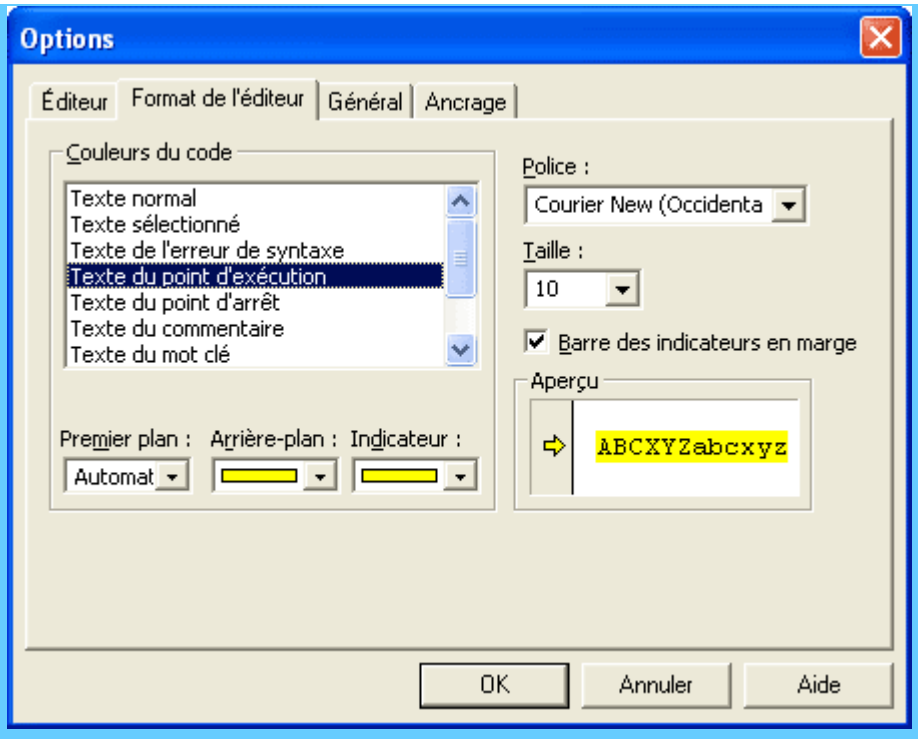


Info-bulles automatique :
Indispensable lors d'un déboguage pas à pas. Elle permet l'affichage de vos variables.

Retrait automatique :
Permet à VBE de placer chaque ligne de code au même niveau que la ligne précédente. Le retrait de lignes se fait par les touches "Tab" et "Shift+Tab". Cette option est nécessaire pour une bonne lecture du code VBA.

Paramètres de la fenêtre :
Les 3 options sont intéressantes. L'édition de texte par glisser-déplacer permet de déplacer à l'aide de la souris le bloc de code sélectionné, l'affichage du module complet par défaut permet l'affichage de toutes les procédures d'un même module et la séparation des procédures oblige VBE à créer des traits entre chaque procédures.

2 - Onglet Format de l'éditeur :



Cet onglet permet de changer la police et son format pour les différentes parties du code inscrit dans vos modules. Je ne vous conseille pas de changer les paramètres par défaut.

Ces options vous permettent de configurer à votre convenance l'éditeur de macros. Si vous débutez, je vous conseille cependant de garder les options par défaut.

Cours VBA - L'enregistreur de macros -

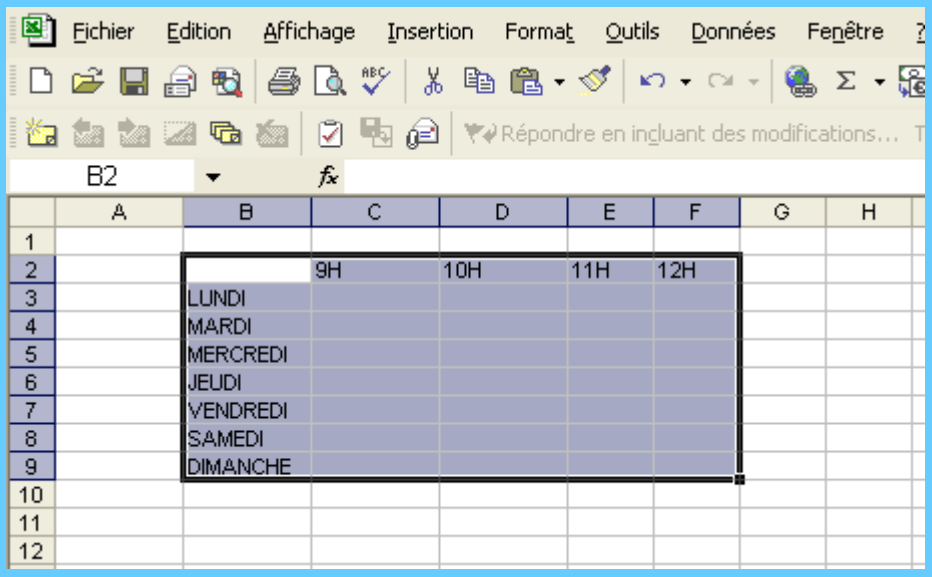
Excel, comme Word ou PowerPoint possède un outil génial : l'enregistreur de macros. Il crée une macro et transforme en langage VBA toutes les commandes effectuées par l'utilisateur dans l'application hôte. Il permet d'automatiser sans aucune connaissance de la programmation certaines de vos tâches et également de se familiariser avec le langage VBA.

Utilisation de l'enregistreur de macros

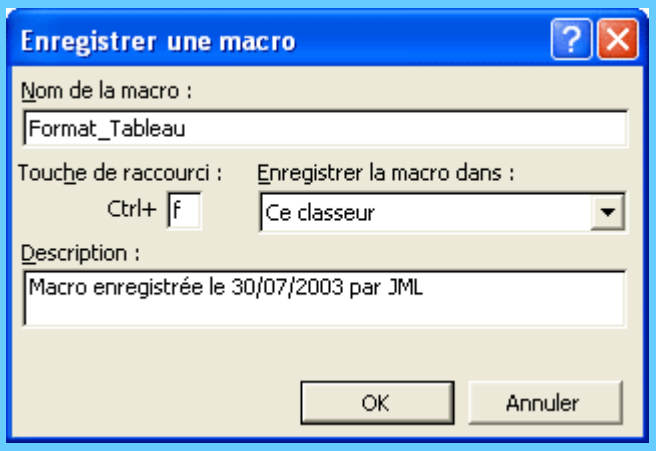
Prenons un exemple :

Je veux mettre au même format tous les tableaux que je crée. Plutôt que de reproduire à chaque fois les mêmes commandes, je vais utiliser l'enregistreur de macros.

- Je sélectionne d'abord mon tableau. En effet, si je le sélectionne après avoir lancé l'enregistrement, la macro reproduirait cette sélection à chaque lancement et s'exécuterait toujours sur les mêmes cellules.



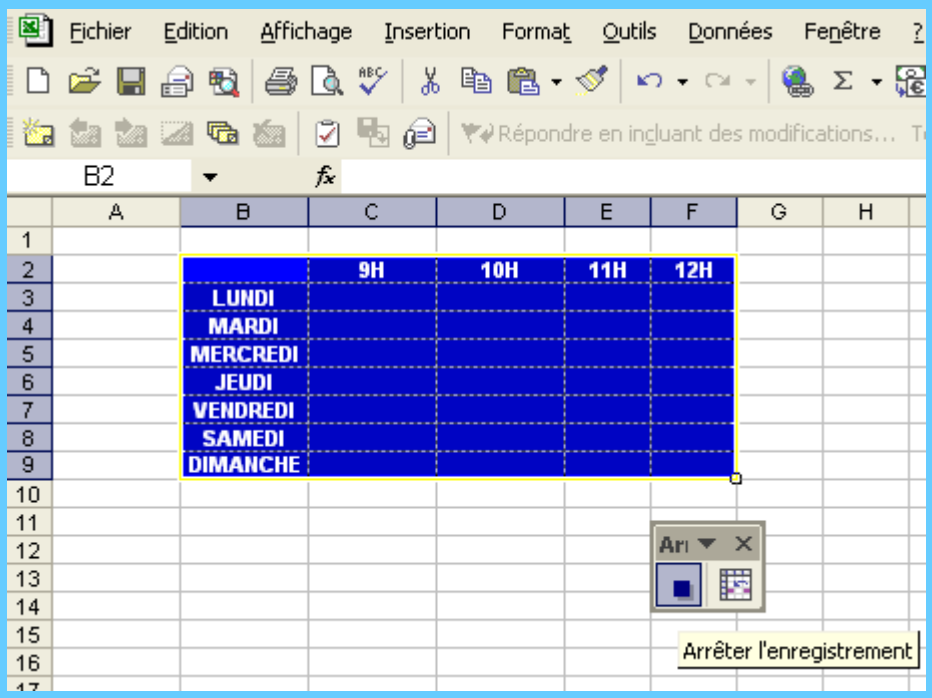
- Je lance l'enregistrement par le menu "Outils - Macro - Nouvelle macro" :



Nom de la macro : Il est important de renommer de façon explicite la macro. Le nom de la macro doit commencer par une lettre et ne doit pas contenir d'espaces. Utilisez le caractère de soulignement pour séparer les mots.
Touche de raccourci : Il est possible de créer un raccourci clavier pour lancer la macro en saisissant une lettre. Vous pouvez utiliser les majuscules et lancer la macro par les touches Ctrl+MAJ+Lettre.
Enregistrer la macro dans : Classeur contenant la macro. La macro ne pourra ensuite s'exécuter que si le classeur dans lequel la macro a été enregistrée est ouvert. Si vous choisissez "classeur de macros personnelles", un nouveau classeur

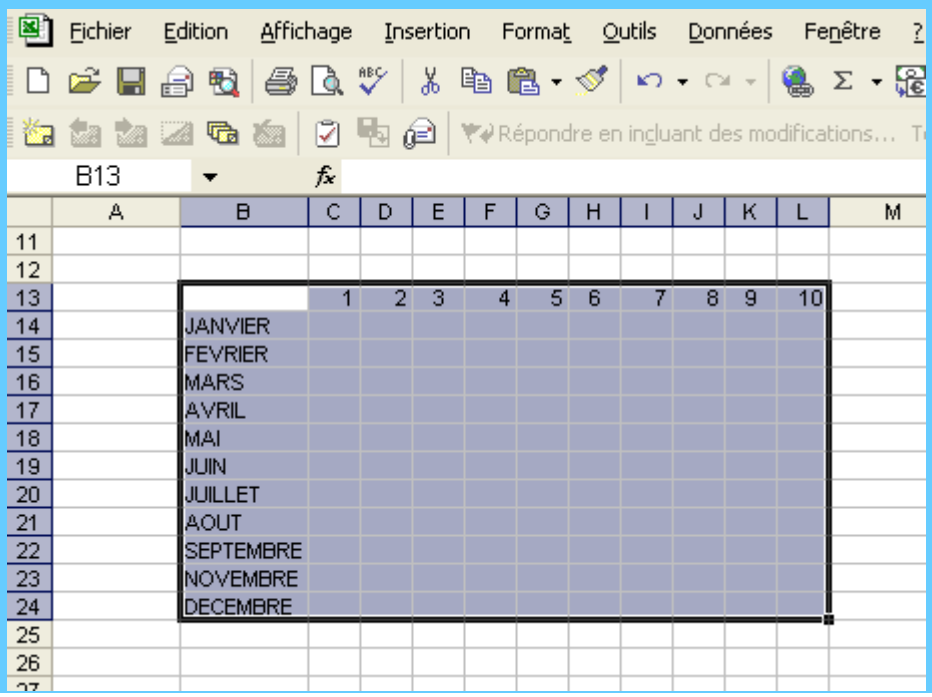
est créé et enregistré dans le dossier "xlstart" de façon que vos macros soient disponibles à chaque fois que vous utilisez Excel.
Description : Vous pouvez donner une description à votre macro.

- Après avoir cliqué sur "OK", la macro va enregistrer toutes les commandes que je vais utiliser. Ici, je change le format de mon tableau (Couleur, Polices, Encadrement).

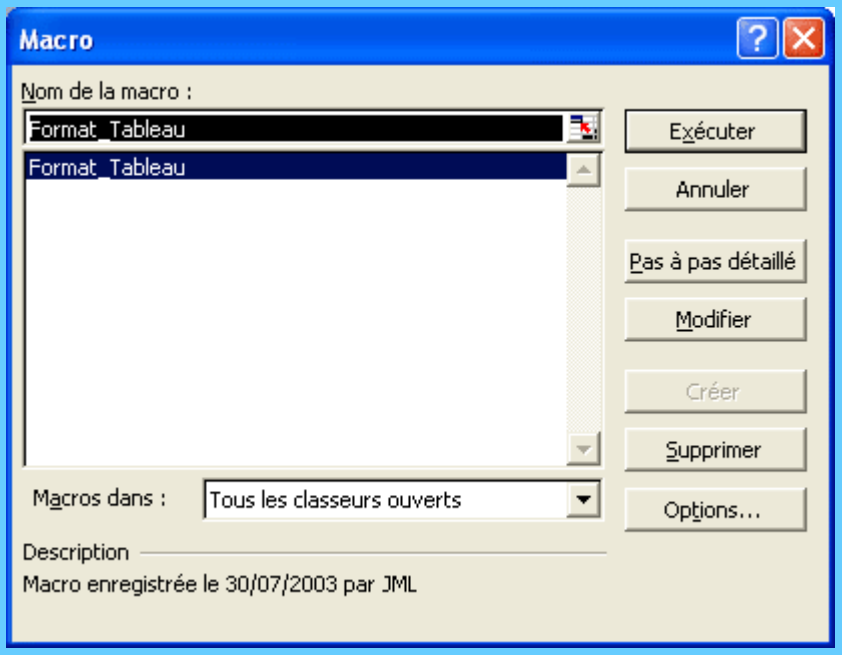


- J'arrête ensuite l'enregistrement en cliquant sur le bouton "Arrêter l'enregistrement" ou par le menu "Outil-Macro-Arrêter l'enregistrement". La macro est créée.

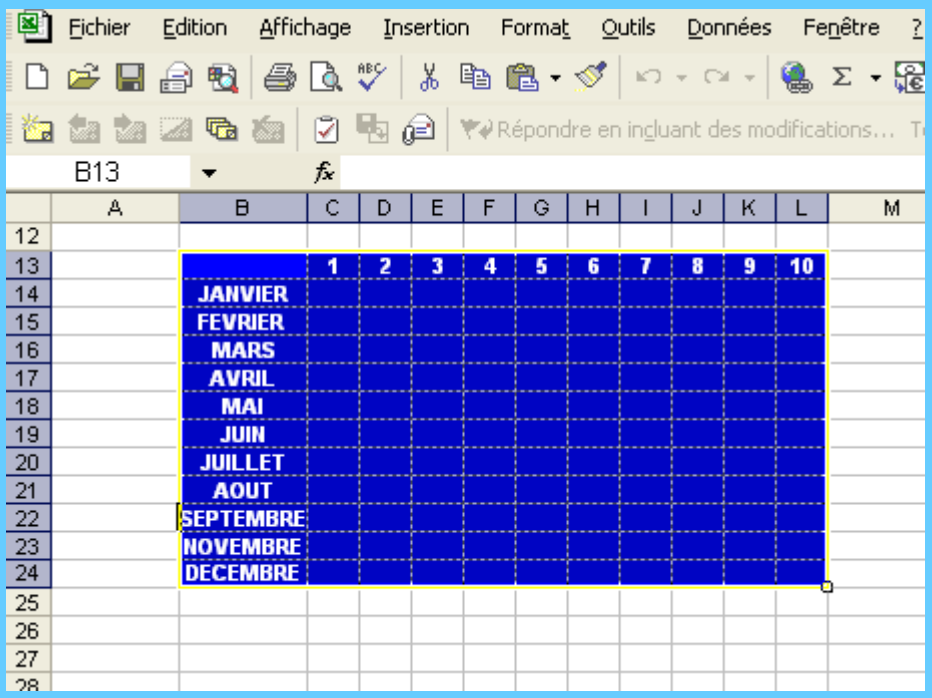
- J'ai un nouveau tableau sur lequel je veux reproduire le même format :



- Je sélectionne mon nouveau tableau et je vais utiliser la macro précédemment créée (à condition que le classeur dans laquelle elle a été enregistrée soit ouvert). Je la lance par le menu "Outils-Macro-Macros" :



- Je sélectionne la macro désirée puis je clique sur "Exécuter" :



- J'aurai également pu lancer la macro par le raccourci clavier "Ctrl+f" que j'avais défini lors de sa création.

L'enregistreur de macros est donc un outil très pratique.
Quelques précautions à prendre lors de son utilisation :

-Bien penser aux commandes que l'on veut enregistrer pour éviter de créer du code inutile.
-Savoir dans quel classeur enregistrer la macro pour son utilisation future.
-Renommer les macros de façon explicite.

Limites de l'enregistreur de macros :

Il ne peut que traduire les commandes réalisées par l'utilisateur dans l'application hôte et écrit en général plus de lignes de code que nécessaire.
Lors de l'écriture de macros, nous verrons comment utiliser et réduire le code crée par l'enregistreur.

Les macros créées sont visibles et modifiables via l'éditeur de macro VBE (Visual Basic Editor).