UFRSTAPS Toulouse

UE11

Projet de développement VBA

MASTER 2

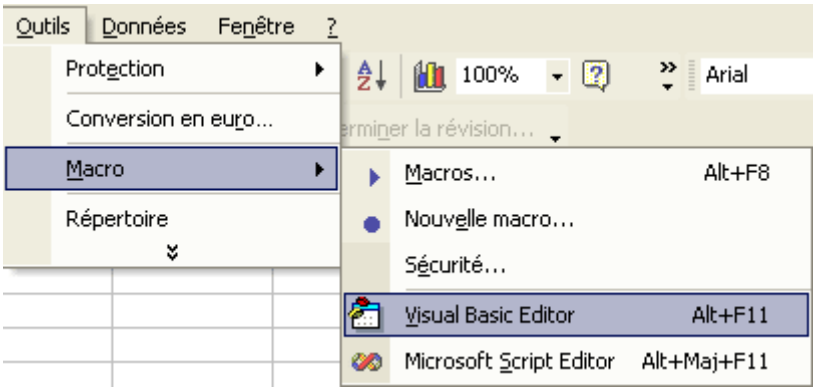
Tribet Hervé

2009

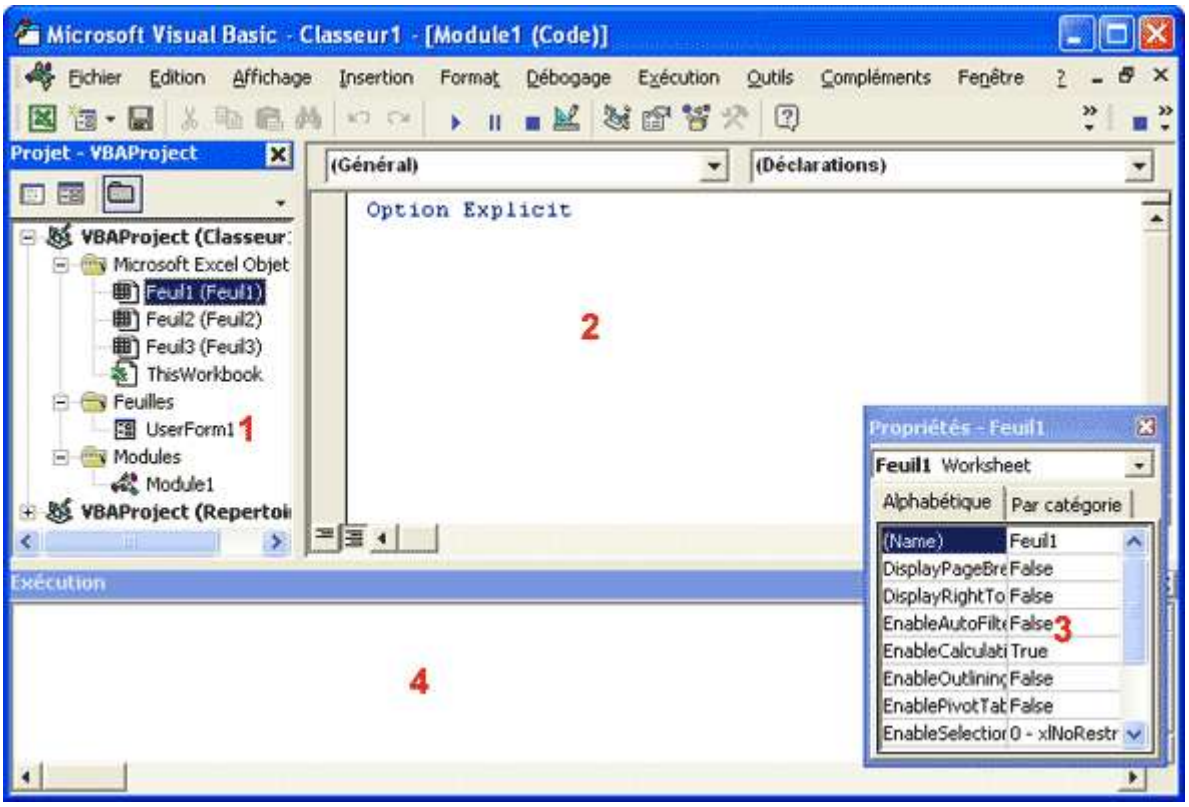
I. Présentation

Visual **B**asic pour **A**pplications (VBA) est le langage de programmation des applications de Microsoft Office. VBA permet d'automatiser les tâches, de créer des applications complètes, de sécuriser vos saisies et vos documents, de créer de nouveaux menus et de nouvelles fonctions pour booster efficacement votre logiciel.

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".



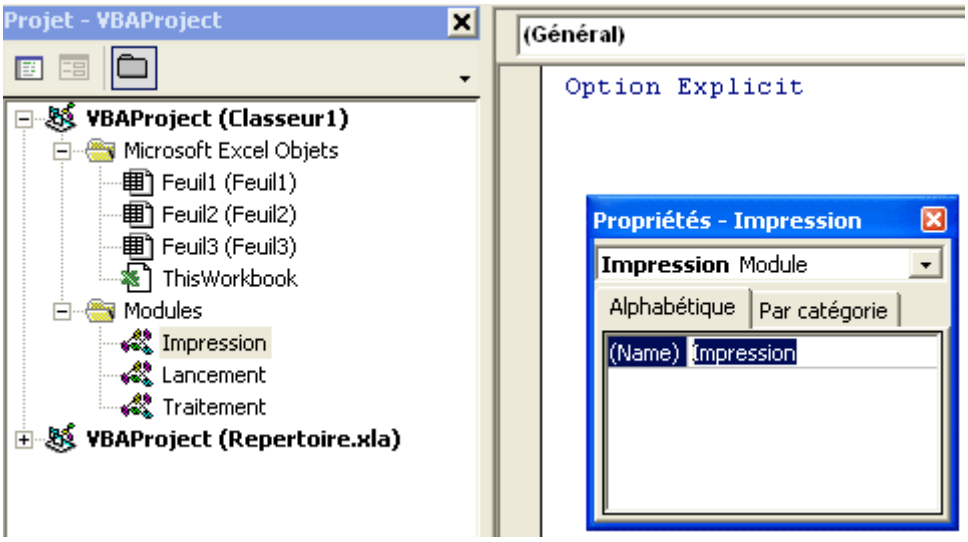
II. VBE



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.

III. Code VBA

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"
```

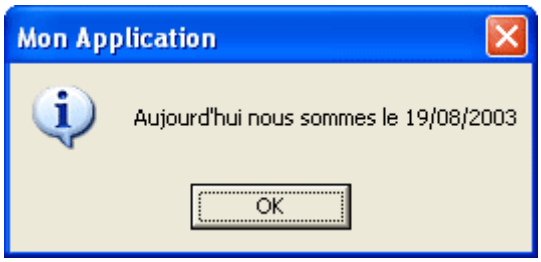
```
    'Message bonjour à l'utilisateur
    MsgBox "Bonjour " & Invite
End Sub
```

Résultat :

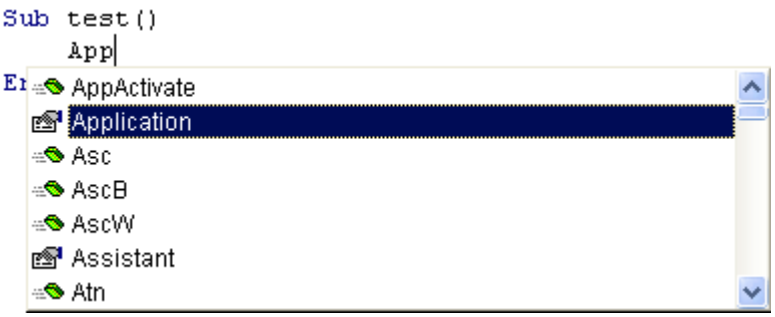
Il n'y a pas de limite de caractères pour chaque ligne d'instruction. Il est toutefois possible d'écrire une instruction sur plusieurs lignes afin d'augmenter la visibilité du code. Pour cela, il faut ajouter le caractère de soulignement avant le passage à la ligne (touche Entrée) :

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

Résultat :



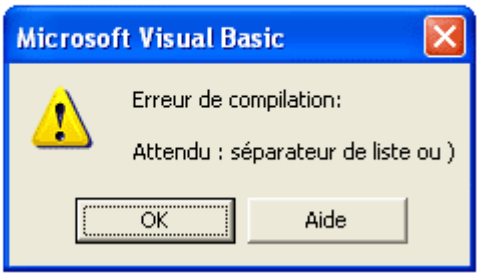
L'option "Info express automatique" permet d'afficher les informations de la fonction que vous venez de taper. Il est également possible d'obtenir de l'aide à tout moment par la combinaison de touches Crtl+j :



La vérification automatique de la syntaxe vous alerte s'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 surprenants résultats.

IV. Vocabulaire VBA

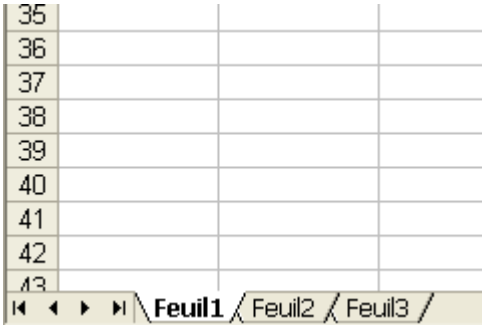
VBA manipule les objets de l'application hôte. Chaque objet possède des propriétés et des méthodes.

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
```

V. Les messages VBA

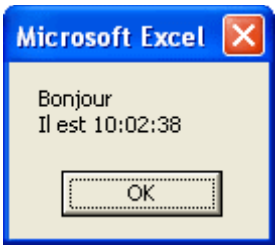
Lors d'une procédure, les messages servent à communiquer avec l'utilisateur.
Il existe des messages qui donnent de l'information et d'autres qui en demandent.

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
```

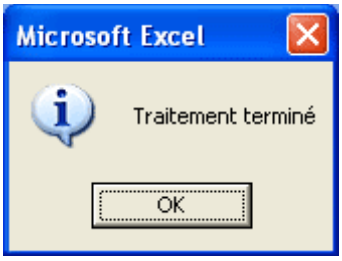


Vous pouvez ajouter une icône concernant le type de message à afficher.
Les types d'attribut icône :

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

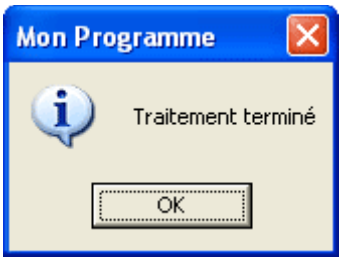
La syntaxe pour ajouter une icône est MsgBox "Message", attribut icône :

```
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"
```

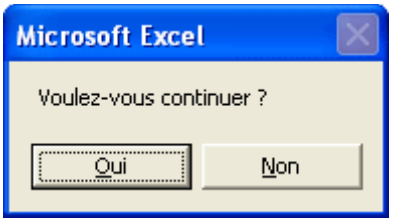


Les MsgBox peuvent également demander une information à l'utilisateur. Dans ce cas, la boite de message comprend plusieurs boutons
Les types d'attribut Boutons :

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

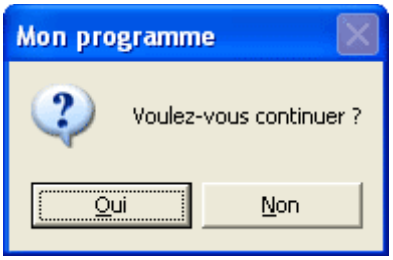
La syntaxe est MsgBox ("Message", attribut bouton):

```
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

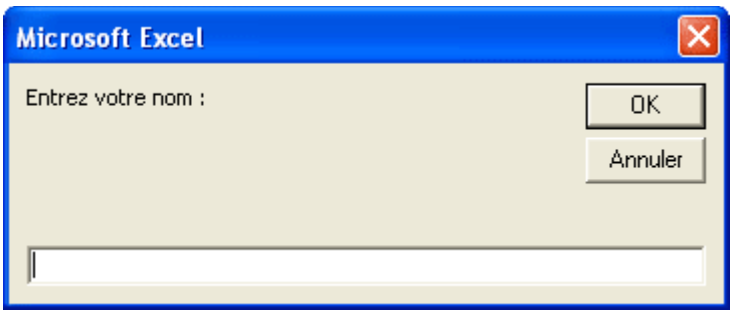
Ainsi, si l'utilisateur clique sur le bouton "OK", MsgBox renvoie la valeur 1, sur le bouton "Annuler" la valeur 2, sur le bouton "Ignorer" la valeur 5 ...
Cette valeur est récupérée dans une variable.

```
'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

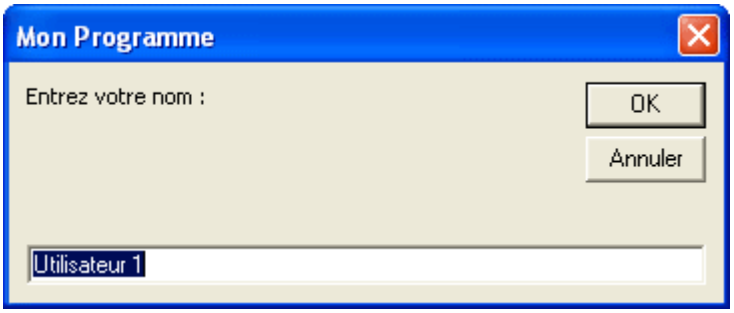
Les InputBox sont des boites de dialogue dans lesquelles l'utilisateur est invité à entrer des données. La syntaxe est : InputBox ("Message").

```
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 ("").

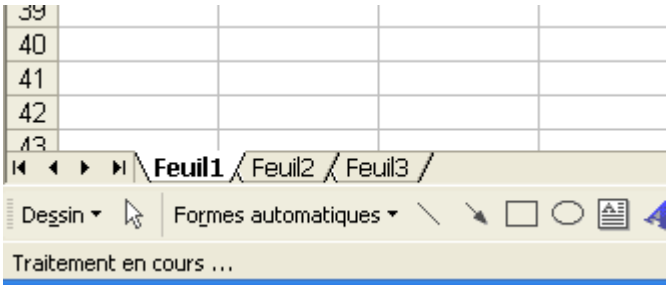
```
Message = InputBox("Entrez votre nom :", "Mon Programme", _
"Utilisateur 1")
```




```
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.

VI. Les variables

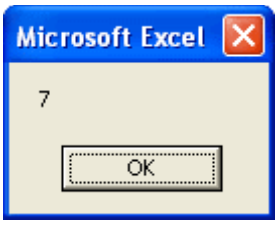
Lors d'une procédure, les variables servent à stocker toutes sortes de données (des valeurs numériques, du texte, des valeurs logiques, des dates ...). Elles peuvent également faire référence à un objet.
Suivant les données que la variable recevra, on lui affectera un type différent. Les différents types 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	**String**	Longueur de la chaîne	1 à 65 400 caractères

fixe			
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\"
```

VII. 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
```

VIII. Les conditions

Les conditions sont très courantes dans les applications VB. Elles peuvent déterminer la valeur que prennent les variables, arrêter une procédure, appeler une procédure, quitter une boucle, atteindre une étiquette.

Les exemples suivants vont déterminer la valeur que prendra la variable Mention par rapport à des notes. Le tableau des notes est :

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

L'instruction la plus courante dans VB est la condition **If** *condition* **Then** *valeur vrai* :

```
'La Note se trouve dans la cellule "A1", la mention
est à mettre dans la cellule "B1"
'Pour trouver la valeur de la mention, on pourrait
écrire :
Dim Note As Integer
Dim Mention As String
Note = Range("A1")
If Note = 0 Then Mention = "Nul"
If Note >= 1 And Note <6 Then Mention = "Moyen"
If Note >= 6 And Note <11 Then Mention = "Passable"
If Note >= 11 And Note <16 Then Mention = "Bien"
If Note >= 16 And Note <20 Then Mention = "Très
Bien"
If Note = 20 Then Mention = "Excellent"
Range("B1") = Mention
```

Si la valeur vraie possède plusieurs lignes d'instructions, la syntaxe devient **If** *Condition* **Then** *Valeur vraie* **End If**.

```
'Si la note est égale à 0, la mention prend comme
valeur "Nul" et la couleur de la police devient
Rouge:
Dim Note As Integer
Dim Mention As String
Note = Range("A1")
If Note = 0 Then
    Mention = "Nul"
    Range("B1").Font.Color = RGB(255, 0, 0)
End If
Range("B1") = Mention
```

Dans notre exemple, l'instruction peut être mieux structurée. La couleur de la police de la mention est rouge si la note est inférieure à 10 et verte si la note est supérieure à 10 en utilisant la syntaxe **If** *condition* **Then** *valeur vrai* **Else** *valeur fausse* **End If**.

```
If Note < 10 Then
    Range("B1").Font.Color = RGB(255, 0, 0)
Else
    Range("B1").Font.Color = RGB(255, 0, 0)
End If
```

Pour calculer la valeur de la mention, on utilisera plus facilement la syntaxe **If** *condition* **Then** *valeur vraie* **ElseIf** *condition* **Then** *valeur vrai* **Else** valeur vraie **End If** en ajoutant autant de fois que nécessaire l'instruction **ElseIf.**

```
Dim Note As Integer
Dim Mention As String
Note = Range("A1")
If Note = 0 Then
    Mention = "Nul"
ElseIf Note >= 1 And Note <6 Then
    Mention = "Moyen"
ElseIf Note >= 6 And Note <11 Then
    Mention = "Passable"
ElseIf Note >= 11 And Note <16 Then
    Mention = "Bien"
ElseIf Note >= 16 And Note <20 Then
    Mention = "Très Bien"
Else
    Mention = "Excellent"
End If
Range("B1") = Mention
```

Dans le cas de conditions multiples, comme dans notre exemple, on préférera le bloc d'instruction **Select Case** *expression* **Case** *valeur expression* **Case Else End Select**.

```
Dim Note As Integer
Dim Mention As String
Note = Range("A1")
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
Range("B1") = Mention
```

Une condition peut appeler une étiquette. Une étiquette représente un endroit de la procédure. Elle se déclare par un nom suivi du signe ":". Dans l'exemple suivant, si i prend la valeur 10, la procédure va directement à la ligne Msgbox "Fin du programme".

```
        Dim i As Integer
        instructions
        If  i = 10 Then GoTo Fin
        instructions
Fin:
        Msgbox "Fin du programme"
```

Les boucles :

Les boucles le plus souvent utilisés sont les boucles **For** ... **Next.** Elles permettent de répéter un nombre de fois défini un bloc d'instructions. Elles utilisent une variable qui est incrémentée ou décrémentée à chaque répétition.

```
Dim i As Integer
'La boucle suivante va écrire les chiffres de 1 à 10
'dans la plage de cellule "A1:A10". La variable i
's'incrémente de 1 à chaque fois
For i  = 1 To 10
    Range("A1").Offset(i - 1) = i
Next i
```

La variable peut être incrémentée d'une valeur différente de 1 par le mot **Step**.

```
Dim i As Integer, j As Integer
'La boucle suivante va écrire les chiffres pairs
'dans la plage de cellule "A1:A10". La variable i
's'incrémente de 2 à chaque fois
j = 0
For i  = 2 To 20 Step 2
    Range("A1").Offset(j) = i
    j = j + 1
Next i
```

La variable peut également être décrémentée. Dans ce cas, le mot **Step** est obligatoire.

```
Dim i As Integer, j As Integer
'La boucle suivante va écrire les chiffres de 20 à
10
'dans la plage de cellule "A1:A10". La variable i
'se décrémente de 1 à chaque fois
j = 0
For i  = 20 To 10 Step -1
    Range("A1").Offset(j) = i
    j = j + 1
Next i
```

A l'intérieur d'un bloc d'instruction **For Next**, l'instruction **Exit For**, peut quitter la boucle avant que la variable n'est atteint sa dernière valeur. Dans le tableau suivant se trouve une liste d'élèves avec leurs 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

Pour connaître la note de Paul, on pourrait utiliser :

```
Dim i As Integer
Dim NbreEleve As Integer, NoteEleve As integer
Dim Cel As Range
'On affecte la cellule "A1" à la variable Cel
Set Cel = Range("A1")
'La derniere ligne - 1 correspond au nombre d'élèves
NbreEleve = Cel.End(Xldown).Row - 1
For i  = 1 To NbreEleve
    If Cel.Offset(i) = "PAUL" Then
        'On récupère la note
        NoteEleve = Cel.Offset(i, 1)
        'puis on sort de la boucle
        Exit For
    End If
Next i
Msgbox "La note de Paul est " & NoteEleve
```

Pour répéter un bloc d'instructions pour chaque objet d'une collection ou pour chaque élément d'un tableau, on utilisera le bloc d'instruction **For Each** *Objet* **In** *Collection* **Next.**
L'exemple suivant mettra la police de couleur rouge si les notes sont inférieures à 10 et de couleur verte si les notes sont supérieures à10.

```
Dim Cel As Range, Cel2 As Range
'On affecte la plage de cellules "B2:B15"
'à la variable Cel
Set Cel = Range("B2:B15")
'Pour chaque cellule de la plage de cellule
For Each Cel2 In Cel
    If Cel2 < 10 Then
```

```
            'Police de couleur rouge
            Cel2.Font.Color = RGB(0, 255, 0)
        Else
            'Police de couleur verte
            Cel2.Font.Color = RGB(255, 0, 0)
        End If
    Next
```

On peut également utiliser l'instruction **Exit For** pour sortir d'un bloc d'instruction **For Each ... Next**.

Les boucles conditionnelles:
Les boucles **While** *condition* **Wend** exécutent un bloc d'instruction tout pendant que la condition est vraie.

```
Dim Calcul As Integer, Compteur As Integer
Compteur = 1
'Le bloc d'instruction suivant va additionner les
' nombres de 1 à 10 (1+2+3+4+5+6+7+8+9+10).
'Tant que la valeur de Compteur est inférieur 11
While Compteur < 11
    Calcul = Calcul + Compteur
    'Ne pas oublier d'incrémenter le compteur sinon
    'la boucle ne pourra pas s'arrêter.
    Compteur = Compteur + 1
Wend
Msgbox Calcul
```

Les boucles **Do Loop** sont mieux structurées que les boucles **While Wend**. On peut à tout moment sortir d'une boucle **Do Loop** par l'instruction **Exit Do**.
La boucle **Do While** *condition* **Loop** exécute un bloc d'instruction tout pendant que la condition est vraie. Dans l'exemple suivant, on veut ajouter l'élève Annie à la liste des élèves.

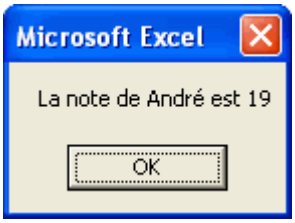
```
Dim Compteur As Integer
Dim Cel As Range
'On affecte la cellule "A1" à la variable Cel
Set Cel = Range("A1")
Compteur = 1
'Le bloc d'instruction suivant va se répéter
'tant que la cellule n'est pas vide
Do While Cel.Offset(Compteur) <> ""
    'Ne pas oublier d'incrémenter le compteur sinon
    'la boucle ne pourra pas s'arrêter.
    Compteur = Compteur + 1
Loop
Cel.Offset(Compteur) = "ANNIE"
```

Dans l'exemple précédent, la condition est testée à l'entrée de la boucle. Dans la boucle **Do Loop While** *condition*, le bloc d'instruction est exécuté une fois avant que la condition soit testée.

```
Dim Compteur As Integer
Dim Cel As Range
'On affecte la cellule "A1" à la variable Cel
Set Cel = Range("A1")
Compteur = 1
'Le bloc d'instruction suivant va se répéter
'tant que la cellule n'est pas vide
Do
    'Ne pas oublier d'incrémenter le compteur sinon
    'la boucle ne pourra pas s'arrêter.
    Compteur = Compteur + 1
Loop While Cel.Offset(Compteur) <> ""
Cel.Offset(Compteur) = "ANNIE"
```

Pour sortir d'une boucle, on utilise l'instruction **Exit Do**. Pour recherche la note de André, on pourrait utiliser :

```
Dim Compteur As Integer, NoteEleve As integer
Dim Cel As Range
'On affecte la cellule "A1" à la variable Cel
Set Cel = Range("A1")
Compteur = 1
'Le bloc d'instruction suivant va se répéter
'tant que la cellule n'est pas vide
Do While Cel.Offset(Compteur) <> ""
    'Si la valeur de la cellule est "ANDRE", on sort
    'de la boucle
    If Cel.Offset(Compteur) = "ANDRE" Then
        Exit Do
    End If
    'Ne pas oublier d'incrémenter le compteur sinon
    'la boucle ne pourra pas s'arrêter.
    Compteur = Compteur + 1
Loop
NoteEleve = Cel.Offset(Compteur, 1)
Msgbox "La note de André est " & NoteEleve
```



Les boucles **Do Until** sont identiques aux boucles **Do While**, seulement le bloc d'instruction est répété tout pendant que la condition n'est pas vraie. La syntaxe est exactement la même, il y a juste le mot **Until** qui remplace le mot **While.** Si on reprend l'exemple précédent, la procédure deviendrait :

```
Dim Compteur As Integer, NoteEleve As integer
Dim Cel As Range
'On affecte la cellule "A1" à la variable Cel
Set Cel = Range("A1")
Compteur = 1
'Le bloc d'instruction suivant va se répéter
'tant que la cellule n'est pas vide
Do Until Cel.Offset(Compteur) = "ANDRE"
    'Ne pas oublier d'incrémenter le compteur sinon
    'la boucle ne pourra pas s'arrêter.
    Compteur = Compteur + 1
Loop
NoteEleve = Cel.Offset(Compteur, 1)
Msgbox "La note de André est " & NoteEleve
```
