Université Paris X-Nanterre
Jean VICTOR

VBA

Chapitre 1 : Les macros

A) Notion de macro

Automatisation des opérations sur Excel, en particulier des opérations ne pouvant faire l'objet d'un calcul.
Par exemple, ouvrir un classeur, actualiser un tableau croisé dynamique, trier, etc.
Ecriture de programmes complexes basés sur Excel.
Nécessité de bien connaître le fonctionnement d'Excel pour définir les macros.
Le langage est le même pour Word et PowerPoint. Mais l'utilisation est moins fréquente

B) Le langage macro Visual Basic for Application (VBA) de Excel

Langage structuré (cf. Algorithmique de base infra) orienté objet. Il est conseillé de lire ce chapitre dès maintenant si vous n'avez pas d'expérience de programmation.
L'écriture de la macro s'effectue principalement par répétition d'une série d'actions "apprises" par le logiciel et traduites dans son langage.
Possibilité de correction et d'adaptation. Ecriture de programmes complets.

C) Ecriture automatique de macros simples

Exemple : effacer l'intégralité d'une cellule (contenu et mise en forme)
1. Lancer l'enregistrement de la macro : **OUTILS, Macro**, **Nouvelle macro**
2. Définir le nom de la macro (sans espace, ex : Ma_macro) et choisir le raccourci clavier
3. Effectuer manuellement l'opération demandée : **EDITION**, **Effacer**, **Tout**
4. Terminer l'enregistrement de la macro : **OUTILS, enregistrer une macro**, **arrêter l'enregistrement**

(Une barre d'outils spéciale, arrêt de l'enregistrement peut apparaître et remplacer utilement cette manœuvre. Si elle n'apparaît pas, la sélectionner dans les barres d'outils. Elle se ferme automatiquement à la fin de l'enregistrement de la macro.)

5. Pour ajouter à une macro existante une autre partie enregistrée automatiquement, la seule solution avec Excel est d'enregistrer une nouvelle macro, puis d'extraire le code (ou programme) intéressant et le réinsérer dans la précédente macro. Il n'est plus possible comme sous Excel 5 de compléter directement la macro par un enregistrement automatique.

VBA a enregistré l'action dans le "projet" qui correspond à la feuille de calcul. La macro enregistrée se trouve dans le dossier Module, généralement sous le nom Module1. Pour retrouver cette macro, **OUTILS, Macro**, **Visual Basic** Editor ou **ALT F11.** Puis choisir : **OUTILS, Macro** dans le nouveau projet appartenant à votre classeur.

```
Sub Ma_macro()
'
' Macro1 Macro
' Macro enregistrée le date par nom de l'utilisateur
    Selection.Clear
End Sub
```

Les informations se trouvant derrière l'apostrophe (') sont des commentaires qui servent uniquement à renseigner le concepteur ou le correcteur sur les contenus de la macro. Ils peuvent êtres supprimés sans dommage. Mais il est pratique de les utiliser pour une correction ultérieure.

D) La structure d'une procédure macro

Sub et End Sub encadrent la macro. Le nom de la macro est suivi d'une parenthèse vide.
(Cette parenthèse sert à transmettre des informations d'une procédure à une autre, cf. Chapitre 4)

Une ligne de programme comprend :

un objet	*un point de séparation*	*une action*
Selection	.	Clear

ou

un objet	*un point*	*une propriété ou qualité*	*un signe =*	*une valeur*
Selection.Police	.	Name	=	"Arial"

Cette seconde ligne permet de définir la police utilisée dans les cellules sélectionnées.
End Sub marque la fin de la procédure si une instruction Exit sub n'a pas été rencontrée auparavant.

Les objets
Les actions portent sur des objets manipulables : une plage de cellules, une feuille de calcul, la bordure d'une cellule, un graphique, un bouton, une barre d'outils, un classeur, etc.

Un objet fait partie d'une collection d'objets de même nature. Les collections sont donc toujours au pluriel. Chacun des objets qui la composent est accessible soit par son nom (nom de la feuille par ex.), soit par son indice ou *item*, (l'énième feuille). Pour désigner un objet, il est souvent nécessaire de donner l'ensemble de sa définition, puisqu'un objet peut être contenu dans un autre objet :

Illustration : M. Pierre Létudiant est... un objet faisant partie de la collection des "Etudiants de l'Université Paris X", partie de la collection des "hommes", sous-partie des "humains", etc. Son nom et son prénom complètent la définition de cet "objet".

L'écriture d'un objet va de gauche à droite, du général au particulier en séparant les collections par des points.

Pour le désigner parfaitement il faut donc écrire : Humains.Hommes.Etudiants_UPX.Noms("Létudiant")

Une cellule est ainsi contenue dans une feuille, elle-même dépendante d'un classeur, etc.…

Par exemple : pour atteindre une plage de cellules ("B33:B34") dans une feuille ("Seconde") du classeur ("Expert.xls") il faut écrire :

```
Workbooks("Expert.xls").Sheets("Seconde").Range("B33:B34").Select
```

Si la macro est lancée directement dans le classeur, il n'est pas nécessaire de le préciser, il est pris par défaut. De même si l'on se trouve dans la feuille ("Seconde"). Dans ce cas, l'écriture suivante est largement suffisante :

```
Range("B33:B34").Select
```

Néanmoins, l'indication de la totalité de la définition de l'objet permet de s'assurer que l'action se déroulera bien là où elle est prévue.

Programmer en langage objet consiste donc à appliquer des méthodes sur des objets pour modifier

leurs propriétés de manière à les rendre conformes aux souhaits du programmeur.

Dans la première macro ci-dessus, Selection désigne l'objet et Clear est la méthode utilisée sur lui.
Selection simplifie avantageusement la désignation complète de la plage sélectionnée.

Les objets, leurs méthodes et leurs propriétés peuvent être consultés sur l'aide en ligne (*F1).* Elle permet surtout de vérifier l'exactitude d'un couplage entre l'objet et sa propriété ou d'ajouter de certaines lignes qui ne peuvent être apprises par le logiciel lui-même.
Les fonctions de VBA peuvent également être obtenues par ce biais.

E) Les méthodes

Elles permettent d'agir sur l'objet auquel elles se rapportent.

```
Sub Lesméthodes()
    Windows("Monclasseur.xls").Activate
    Sheets("Mafeuille").Select
    Range("B1").Select
    Selection.Copy
    Range("C1").Select
    Activesheet.Paste
    Range("D1").Clear
    ActiveWorkbook.Save
End Sub
```

Activer (Activate) est une méthode qui rend actif le classeur
Utile pour être certain de travailler sur la bonne feuille
La plage est un objet, **Sélectionner** (Select) une méthode.
La sélection est copiée, **Copier** (Copy) est une méthode
Changement de cellule.
Coller (Paste) est une méthode.
Effacer (Clear) est effectué ici directement sans sélection préalable
Enregistrer (Save) est une méthode de l'objet classeur.

F) Les propriétés

La même opération (copier/coller) pourrait être réalisée à partir des propriétés.

```
Range("C1").Value = Range("B1").Value
```

Value est une propriété de la plage B1 qui est transmise à la plage C1

Les méthodes ont souvent (mais pas uniquement) pour effet de modifier les propriétés.

Exemple de macro d'encadrement d'une cellule avec des épaisseurs et des couleurs différentes.

```
Sub Encadrement()
    Selection.Borders(xlLeft).Weight = xlMedium
    Selection.Borders(xlLeft).ColorIndex = 2
    Selection.Borders(xlRight).LineStyle = xlNone
    With Selection.Borders(xlTop)
        .Weight = xlThin
        .ColorIndex = 3
    End With
```

Commentaires

Définition d'une propriété de l'objet "bordure gauche de la sélection".
Idem pour bordure droite.

With permet d'éviter de répéter le nom de l'objet. Les propriétés sont présentées en ligne, précédées d'un point jusqu'à End With.

```
    With Selection.Borders(xlBas)
        .Weight = xlThick
        .ColorIndex = 4
    End With
    Selection.Contour LineStyle:=xlNone
End Sub
```

Epaisseur est une propriété.
Tout comme la couleur définie par un numéro d'ordre dans la palette.

Les indications commençant par xl comme xlLeft, xlRight, etc. sont des constantes de Excel dans VBA.
Elles sont écrites par le logiciel lui-même lors de l'enregistrement. Il n'est donc pas nécessaire de les mémoriser.

G) Le lancement de la macro

Pour utiliser la macro il faut l'appeler. Plusieurs techniques sont possibles.
1. Raccourci clavier choisi au moment de la création de la macro.
2. Exécuter la macro par **OUTILS**, **Macro**, nom de la macro et Exécuter.
3. Affecter la macro à un bouton d'une barre d'outils.
   AFFICHAGE, Barres Outils, Personnaliser, Commandes, Macros, Elément de menu personnalisé ou un bouton. Porter l'un de ces deux éléments dans la barre d'outils ou dans le menu, puis bouton droit de la souris, modifier le nom si nécessaire et enfin lui affecter la macro.
   Cette opération peut être réalisée pour toute barre d'outils qu'elle ait été créée par l'utilisateur ou qu'elle soit une barre standard. Les barres d'outils et les articles de menus (deux variantes pour un même concept) sont attachés au classeur pour lequel a été créée la macro. Lorsque le classeur est fermé, les outils personnalisés disparaissent également.
4. Affecter la macro à un objet dessiné dans la feuille
   choisir la barre dessin puis un objet, cliquer sur le bouton droit et lui affecter la macro.
5. D'autres méthodes (macro automatiques sont également possibles, cf. chapitre 4).

H) Arrêter une macro

Généralement, il suffit d'utiliser la touche Echappement (Escape) pour arrêter la macro. Dans le pire des cas (boucle infinie), il faut "mettre fin à la tâche" en utilisant Ctrl, Alt, Suppr. Par prudence, enregistrer les données et le programme avant de lancer une macro contenant des boucles non testées.

I) Développer la macro par apprentissage

La macro n'est généralement pas parfaite du premier coup. Pour ajouter des opérations à une macro existante deux solutions sont possibles :
- soit écrire manuellement la suite avec des éléments algorithmiques (cf. chapitre 2)
- soit faire écrire une nouvelle macro puis en extraire les éléments intéressants et les insérer dans l'ancienne procédure

J) Affectation de valeurs ou de formules dans des cellules

a) Adressage absolu

Les opérations auront toujours lieu dans les mêmes cellules.

```
Sub Absolue()
    Range("A1").Select
    ActiveCell.Formula = "Début de la feuille"
    Range("A2").Select
    ActiveCell.Formula = "Ligne suivante"
    Range("A3").Select
End Sub
```

```
Sub Absolue2() 'sans sélection

    Range("A1").Formula = "Début de la feuille"
    Range("A2").Formula = "Ligne suivante"
    Range("A3").Select
End Sub
```

L'objet de l'action est la cellule A1, élément de l'objet Range (Plage). Une fois sélectionnée, elle devient la cellule active (ActiveCell) parce qu'elle est seule. Sinon, ce serait une Selection.
Formula est la propriété définie par la ligne de programme. Le texte s'écrit dans la cellule.

Ce type d'écriture est obtenu en cliquant dans la barre Arrêt de l'enregistrement la case Références relatives (commande basculante). Par défaut, l'option est l'écriture absolue. Pour passer en relatif, cliquer sur le bouton qui paraît enfoncé (pas très visible). Durant tout le temps de la session ouverte, le bouton reste enfoncé. Pour plus de sécurité, l'enfoncer à nouveau avant de terminer la macro.

L'autre manière de désigner une cellule de manière absolue est :

```
Cells(ligne, colonne)
```

Elle est plus pratique que Range puisqu'elle ne dépend que de nombres calculables dans un programme.

Pour entrer les dates des 300 jours qui viennent dans la première colonne (n°1), il suffit d'écrire :

```
Sub Troiscentsjours()
    For i = 1 to 300
        Cells(i, 1).Value = Date + i        'attention en anglais les arguments sont séparés par des
    Next                                    'virgules et non par des points virgules comme dans
End Sub                                     'les formules d'Excel.
```

Date est une instruction de VBA qui renvoie la date du système de l'ordinateur.

b) Adressage relatif

Si l'opération doit se produire dans des cellules situées relativement à la cellule de départ, il faut cocher le bouton **références relatives**. Le programme devient :

```
Sub Macro3()
    ActiveCell.FormulaR1C1 = "N'importe où dans la feuille"
    ActiveCell.Offset(1, 0).Formula = "Cellule suivante en dessous"
End Sub
```

La première ligne écrit là où se trouve la cellule sélectionnée.
Le décalage sans sélection permet d'écrire immédiatement dans la cellule en dessous.
Le premier argument représente le nombre de lignes du décalage, le second celui des colonnes.

```
ActiveCell.Offset(10, 2)
```

Cette ligne permet d'agir directement sur la cellule se trouvant dix lignes en dessous et deux colonnes à droite de la cellule active.
La valeur des décalages en ligne ou en colonne peut être négatifs. Il faut cependant ne pas sortir du cadre de la feuille de calcul.
Les valeurs des lignes et des colonnes peuvent être calculées dans une boucle par exemple. Le programme suivant écrit en diagonale les dix premiers nombres. Les numéros de la ligne et de la colonne dépendent de la valeur du compteur de la boucle diminuée de 1 pour bien commencer en A1 ou R1C1.

```
Sub Décalage()
    Cells(1, 1).Select
    For i = 1 à 10
        ActiveCell.Offset(i - 1, i - 1).Value = i
    Next
End Sub
```

Il existe une variante pour désigner des cellules relatives à une cellule de départ en utilisant Range après ActiveCell. *Dans ce cas, le tableur est considéré comme débutant à la cellule active. Elle ne permet donc pas d'aller vers la gauche ou vers le haut !*

```
    ActiveCell.Range("B2").select
```

La cellule active se situe "théoriquement" en A1. L'action ci-dessus désigne donc la cellule qui se trouve une colonne (B) à droite et une ligne (2) en dessous de la cellule active.
Cette adresse peut être calculée en concatènant des adresses mais uniquement sur la colonne :

```
For i = 1 to 10
    ActiveCell.Range("B" & i).select
Next
```

Dans cas, la sélection se décale progressivement de 10 lignes vers le bas. Il est impossible de programmer un mouvement vers la droite, vers la gauche et vers le haut. Il faut donc mieux utiliser la précédente écriture.

K) Lecture des valeurs dans les feuilles de calcul

Lecture de la sélection en cours

```
Lavaleur = ActiveCell.Value 'ou Formula selon le contenu (texte par exemple). On utilise presque toujours Value
```

Lecture après sélection

```
Cells(3, 3).Select
Lavaleur = Selection.Value
```

Lecture directe sans sélection

```
Lavaleur = Cells(3, 3).Value
```

Lecture des cellules environnantes

La valeur d'en dessous, sans sélection

```
Lavaleur = ActiveCell.Offset(1, 0).Value
```

La valeur d'à côté, sans sélection

```
Lavaleur = ActiveCell.Offset(0, 1).Value
```

Lecture d'une valeur nommée dans la feuille de calcul, ici le nom est "équipe", il doit être le seul nom donné dans l'ensemble du classeur. Si ce n'est pas le cas, il faut préciser la feuille concernée. Noter que cette écriture ne nécessite pas de guillemet autour du nom de la plage.

```
Lavaleur = [Equipe].Value
Lavaleur = [France!Equipe].Value
```

Lecture d'une série finie de valeurs à partir de la première sans sélection

```
For i = 1 à 9
    Lavaleur = Cells(2, 3).Offset(i, 0).Value
Next
```

Lecture d'une série variable de valeurs à partir de la première

```
Cells(2, 3).Select
While Non (IsEmpty (Selection.Value))        'tant que la cellule n'est pas vide
    ActiveCell.Offset(1, 0).Select
    Lavaleur = ActiveCell.Value
Wend
```

La lecture d'une valeur n'entraîne pas son utilisation immédiate. La variable recueillant l'information pourra donc être travaillée jusqu'à son utilisation dans une feuille de calcul.

```
For i = 1 à 9
    Lavaleur = Cells(2, 3).Offset(i, i).Value
    Lasomme = Lasomme + Lavaleur
Next
Msgbox Lasomme
```

Ce programme permet de calculer la somme de la diagonale de la cellule L3C4 à L11C12, formule difficile à écrire dans une feuille de calcul (sauf à être astucieux[1]).

Avec l'ensemble de ces indications, il est possible de faire des macros efficaces et assez complexes. Mais il est presque toujours nécessaire pour aller plus loin d'utiliser l'algorithmique pour dynamiser et contrôler les opérations.

L) Les sous-programmes

Comme dans tout langage informatique, les opérations répétitives peuvent être concentrées dans un sous-programme qui sera appelé dès que besoin. Ce sous-programme peut (mais ce n'est pas une nécessité) requérir des informations de la macro principale. Ces « arguments » sont passés par l'intermédiaire des parenthèses qui suivent le nom du sous-programme.

[1] Pour réaliser une telle opération, il faudrait sélectionner la diagonale et lui donner un nom. Ensuite il suffit d'utiliser la formule =Somme(dunom).

L'argument prend alors le nom de la variable qui lui correspond :

```
Sub principale()
   sp1 ActiveCell
End Sub

Sub sp1(cellule)
   Debug.Print cellule.Value * 2
   sp2 cellule
End Sub

Sub sp2(cellule2)
  Debug.Print cellule2.Value * 4
End Sub
```

Dans ce cas, la procédure principale envoie l'adresse de la cellule active au premier sous-programme qui la traite puis l'envoie à la seconde procédure. On notera que les noms ont peu d'importance (en respectant les normes traditionnelles).
Plusieurs arguments peuvent être passés d'une procédure principale à un sous-programme, il suffit de les séparer par des virgules dans l'appel de la sous procédure. Il doit y avoir autant de valeurs passées que de variables inscrites dans la sous procédure.

```
Sub principale()
   sp1 [A1], [C12], [D5]
End Sub

Sub sp1(cellule1, cellule2, cellule3)
   Debug.Print cellule1.Value * cellule2.Value * cellule3.Value
End Sub
```

M) Les fonctions

Les fonctions ressemblent à des macros mais se lancent différemment. Elles peuvent servir de sous-programme dans une procédure normale ou être directement appelées par Excel comme une fonction personnalisée.
Le principe est qu'une fonction renvoie la dernière valeur attribuée à la variable qui porte le nom de la fonction.

Par exemple, il n'existe pas dans Excel une fonction intégrée qui totalise le nombre de caractères d'une plage de cellules. On créé alors la fonction suivante :

```
Function Compte_Lettres(cellules)
   For Each cel In cellules
      Compte_Lettres = Compte_Lettres + Len(cel.Value)
   Next
End Function
```

La fonction attend toujours un argument qui est passé, comme dans les sous-programmes ordinaires, par l'intermédiaire de la variable entre parenthèse après le nom de la fonction.
Pour utiliser cette fonction, il suffit dans une cellule d'Excel d'écrire

```
= Compte_Lettres(plage de cellules)
```

Les fonctions personnalisées sont propres à un classeur, celui dans lequel se trouve le module VBA. Elles peuvent être retrouvées en choisissant dans les fonctions …. fonctions personnalisées !
Ceci dit, avant de créer des fonctions personnalisées, mieux vaut apprendre à utiliser toutes les fonctions d'Excel, y compris les fonctions complémentaires.

Chapitre 2 : Algorithmique de base

L'intérêt d'une macro est de pouvoir aller au-delà des opérations qui sont habituellement faites par le tableur.
Il faut donc pouvoir écrire des macros manuellement et pas seulement se contenter des macros enregistrées automatiquement.

A) Affectation de valeurs aux variables

a) Affectation simple

Il s'agit de donner une valeur à une variable propre à la procédure. Le nom de la variable ne doit pas comprendre d'espaces (la variable "La valeur" doit être écrite "Lavaleur"). Ne pas utiliser de mot-clé (comme Sub, End, If, etc.)

```
Lavaleur = 10
Lasecondevaleur = Lavaleur * 12
```

b) Saisie d'une valeur au clavier par l'utilisateur

La fonction est : InputBox(prompt, title, default).
Le Prompt est le texte questionnant l'utilisateur. Title est le titre de la boite de dialogue, Default est la valeur par … défaut.
La valeur renvoyée est toujours une chaîne de caractères. Pour la transformer en nombre utiliser Str.
Le résultat peut être affecté à une variable ou traité directement. Ex:

```
réponse = InputBox (prompt:="Quel nom ?", title:="Enregistrement", default:="")
If InputBox (prompt:="Quel nom ?", title:="Enregistrement", default:="Rien") <> "Rien then
```

B) Editer un message

Pour tester des macros, informer l'utilisateur ou lui proposer des options sur la suite du programme

Msgbox "texte du message"	renvoient des informations permettant
ou	de
Msgbox Lavaleurcalculée	Suivre le déroulement.
Msgbox "Poursuivre", vbYesNo… ou vbYesNoCancel	Permet de tester la réponse choisie

```
Sub essai()
    Lavaleur = 10
    Msgbox Lavaleur
    Lasecondevaleur = Lavaleur * 12
    Msgbox Lasecondevaleur
End Sub
```

```
Sub essai2()
    Réponse = Msgbox ("Faut-il continuer", vbYesNoCancel)
    If Réponse = vbYes then
    Else
    If Réponse = vbNo Then
….
```

C) Test

a) Fonction If

Les tests s'écrivent comme dans tout langage structuré à partir de If... Then … Else... End If

```
IF condition Then
    action(s) 1
Else
    action(s) 2
End If
```

Eventuellement sur une ligne si l'action est unique. La lisibilité est cependant moins bonne.

```
IF condition Then action 1 Else action 2
```

Les tests peuvent être imbriqués

```
IF condition1 Then
    IF condition2 Then
        action 1
    Else
        action 2
    End If
Else
    action 3
End If
```

Ex : saisie d'une note testée pour éviter qu'elle ne dépasse 20/20 ou ne soit inférieure à zéro :

```
Sub leTest()
```

```
        Lanote = Str(Inputbox(Prompt:="Quelle note ?", Title:="Test", Default:=-1))
        If Lanote > 20 Then
            Msgbox "Erreur de saisie par excès"
        Else
            If Lanote < 0 Then
                Msgbox "Erreur de saisie par excès"
            Else
                Msgbox "La note est correcte"
            End If
        End If
End Sub
```

b) Fonction Select Case

Cette fonction permet de décomposer un test multiple en autant de "cas" que nécessaire.

```
Select Case Résultat logique d'un test
        Case premier cas
            Action(s) ...
        Case second cas
            Action(s) ...
        Case Else autres cas
            Action(s) ...
End Select
```

```
Sub selonlecas()
    Réponse = Inputbox("Entrez un nombre","Test selon",1)
    Select Case Réponse
        Case Is = 1
            Msgbox "le nombre est un"
        Cas Is = 2
            Msgbox "le nombre est deux"
        Cas Else
            Msgbox "le nombre est supérieur à deux"
    End Select
End Sub
```

c) L'opérateur Like

Il facilite la comparaison des chaînes de caractères. Le principe est de comparer la chaîne à une image. Ex:

```
If Réponse Like "*a*" Then
```

Si la réponse est Maîtrise alors la réponse est vraie car elle contient bien un "a".

Les images sont composées de caractères précis et de signes génériques. Les principaux sont les suivants :

?	remplace un caractère	(ex : "UPX" comme "UP?" renvoie vrai)
*	remplace toute chaîne de caractères	(ex :"UPX" comme "*X" renvoie vrai)
#	remplace un nombre quelconque	(ex: "1996" comme "199#" renvoie vrai)
[A-Z]	liste auquel doit appartenir le caractère testé	(ex: "U" comme [P-V] renvoie vrai)
[!A-Z]	liste auquel ne doit pas appartenir le caractère testé	(ex: "U" comme [!P-V] renvoie faux)

Pour des exemples et des solutions plus complexes, consulter l'aide en ligne sur Like.

D) Répétition

a) La répétition contrôlée d'une action

Elle est basée sur la syntaxe suivante

```
For compteur = valeur initiale To valeur finale Step lepas      ' le pas est optionnel
    actions
Next compteur                                                   ' le rappel du compteur est optionnel
```

Ex: Afficher une série de 10 nombres positifs

```
Sub Compteur1()
    For i = 2 to 20 Step 2
        Msgbox i
    Next i
End Sub
```

Les boucles peuvent être imbriquées à condition de respecter l'ordre et la dénomination des compteurs.

```
Sub Compteur2()
'Les résultats de ces deux boucles
'se trouvent ci-contre
    For i = 10 to 1 Step -2
        For J = 2 to 20 Step 2
            Msgbox i *j
```

Série 1	Série 2	Série 3	Série 4	Série 5
20	16	12	8	4
40	32	24	16	8
60	48	36	24	12
80	64	48	32	16
100	80	60	40	20

```
        Next j
    Next i
End Sub
```

120	96	72	48	24
140	112	84	56	28
160	128	96	64	32
180	144	108	72	36
200	160	120	80	40

b) Répétition conditionnelle

Elle est basée sur l'algorithme While... Wend

```
Sub Boucleconditionnelle()
    ActiveWorkbook.save                 'prudence
    i = 2
    While i =< 20                   '<= ne fonctionne pas
        Msgbox i
        i = i + 2                       ' Ici placé pour que la valeur 2 apparaisse
    Wend
End Sub
```

Cette boucle est très utile lorsque le nombre des répétitions à effectuer n'est pas connu. Cependant, elle peut n'avoir aucune fin. Dans ce cas, il faut "planter" le programme pour l'arrêter… et le programme est perdu s'il n'a pas été sauvegardé auparavant. Vérifier la possibilité d'en sortir et ne pas oublier la première ligne de programme qui sauvegarde le programme avant de se lancer.

c) La boucle For Each

Le nombre des objets à travailler n'est pas toujours connu et il n'est pas toujours facile d'en connaître le nombre. Recourir aux boucles conditionnelles suppose de savoir compter le nombre des objets en cause.
La boucle For Each permet de résoudre ces problèmes avec une grande facilité. Elle doit donc être privilégiée en permanence.
Sa structure est la suivante :

```
For Each nom_objet_considére_individuellement in collection_des_objets
'Traitement de chacun des objets de la collection grâce au nom fixé par le programmeur
Next
```

Par exemple pour saisir toutes les cellules d'une sélection. La collection est la Selection, chacune des cellules est appelée tour à tour cl. Tout autre nom admissible aurait pu être choisi.

```
For Each cl In Selection
    cl.Value = Application.Trim(cl.Value)
    'supprime les espaces en utilisant la fonction SUPPRESPACE() de Excel désigné ici comme Application
Next
```

Ce programme peut donc fonctionner qu'il y ait une seule cellule dans la sélection ou toute une plage.

Autre exemple qui cache toutes les barres d'outils :

```
For Each cette_barre In Application.CommandBars
    On Error Resume Next
    cette_barre.Visible = False
Next
```

Et pour ne rendre visible que les barres les plus utiles

```
Application.CommandBars("Standard").Visible = True
Application.CommandBars("Formatting").Visible = True
Application.CommandBars("Visual Basic").Visible = True
Application.CommandBars("Drawing").Visible = True
```

Chapitre 3 : Les mots-clés de Visual Basic

Les indications suivantes présentent les principaux mots-clés de Visual Basic. Pour des listes complètes et des indications sur la syntaxe, l'utilité, des exemples d'utilisation se reporter à l'aide en ligne. Un fichier Listevba.xls établit la correspondance entre les termes français (Excel 5) et la version anglaise (Excel 97). Cette liste n'est plus fournie avec les versions suivantes (me la demander en cas de besoin).

Il n'est pas possible de donner la correspondance entre tous les objets et leurs propriétés et toutes les méthodes qui peuvent leur être appliquées. Généralement, l'écriture est automatique puisque le logiciel apprend et transcrit les opérations à réaliser. Un même mot-clé peut désigner à la fois un objet, une propriété et une méthode ! Tout dépend du contexte dans lequel il est utilisé. Pour simplifier, les indications ci-dessous ne reprennent que les affectations principales.

A) Collections et Objets

Ensemble d'objets de même nature (pour distinguer les objets et les collections, ces dernières sont au pluriel). Tous les objets peuvent ainsi être regroupés en collections. (Des collections personnelles peuvent être créées). Les collections sont elles-mêmes des objets !

Principales collections			
Application Excel		Objets dessinés	
Menus	Menus	**Pictures**	Images
ShortcutMenus	Menus Contextuels	**Lines**	Traits
MenuItems	Eléments Menus	**Drawings**	Dessins
MenuBars	Barres Menus	**DrawingObjects**	Objets Dessinés
Toolbars	Barres Outils	**GroupObjects**	Objets Groupés
ToolbarButtons	Boutons Barre Outils	Buttons	Boutons
ScrollBars	Barres Défilement		
Windows	Fenêtres		
Panes	Volets		
Classeurs		Objets des graphiques	
Workbooks	Classeurs	**ChartObjects**	Objets Graphique
Sheets	Feuilles	**Charts**	Graphiques
Worksheets	Feuilles Calcul	**Series**	Série
SelectedSheets	Feuilles Sélectionnées	**SeriesLabels**	Etiquettes Série
Rows	Lignes	**SeriesLines**	Lignes Série
Columns	Colonnes	**Trendlines**	Courbes Tendance
Cells	Cellules		
SpecialCells	Cellules Spéciales		
Plages		Outils d'Excel	
Names	Noms		
Comments	Commentaires	**PivotFields**	Champs Dynamiques
Styles	Styles	**PivotTables**	Tableaux Croisés
Borders	Bordures	**Scenarios**	Scénarios
Characters	Caractères		
Colors	Couleurs		

Une opération peut être menée sur l'ensemble de la collection par l'intermédiaire de la fonction Each.

Exemple, pour faire apparaître les feuilles de calcul masquées.

```
Sub collection()
    For Each fc in Sheets   'fc est un nom de variable quelconque
        fc.visible = true
    Next
End Sub
```

Pour certaines collections l'opération peut s'effectuer immédiatement sur l'ensemble de la collection :

```
Application.ActiveSheet.DrawingObjects.Visible = Faux' masque tous les objets dessinés.
```

B) Méthodes

Les "méthodes" permettent d'agir sur des objets. C'est pourquoi leur nom est généralement un verbe exprimant une action. (Les méthodes sont ici regroupées en ligne par affinité). Leur signification est évidente pour qui parle l'anglais courant (la traduction est néanmoins donnée à côté).

Méthodes pour gérer les documents					
• Open	Ouvrir	• Close	Fermer	• Quit	Quitter
• Save	Enregistrer	• SaveAs	Enregistrer Sous		
Méthodes pour gérer les feuilles de calcul					
• Insert	Insérer	• Activate	Activer	• Calculate	Calculer
• Show	Afficher	• Hide	Masquer		
• Protect	Protéger	• Unprotect	Oter Protection		
Méthodes pour gérer les cellules des feuilles de calcul					
• Select	Sélectionner	• Deselect	Désélectionner	• Offset	Décaler
• Copy	Copier	• Paste	Coller	• Cut	Couper
• PasteSpecial	Collage Spécial	• RefreshTable	Actualiser Tableau		
• Group	Grouper	• Ungroup	Dissocier Groupe	• AutoFit	Ajuster Automatiquement
• Find	Rechercher	• Replace	Remplacer	• Sort	Trier
• Delete	Supprimer	• Clear	Effacer	• Cancel	Annuler
• ClearContents	Effacer Contenu	• ClearFormats	Effacer Format		
• CreateNames	Créer Noms	• ApplyNames	Affecter Noms		
• FillDown	Recopier Bas	• FillLeft	Recopier Gauche	• FillRight	Recopier Droite
Méthodes utilisées uniquement pour gérer les macros ou l'environnement de travail					
• AddMenu	Ajouter au Menu	• Add	Ajouter		
• Evaluate	Evaluer	• Goto	Atteindre	• Wait	Attendre

C) Propriétés

Les propriétés sont des qualités des objets. Ils sont donc souvent (mais pas toujours) des adjectifs qualificatifs. Ainsi Afficher est une méthode tandis qu'Affiché est une qualité (une propriété) qui se traduit par vrai ou faux. Définir une propriété est donc aussi une façon d'agir sur un objet.

Afficher une feuille de calcul revient à définir sa propriété comme Affiché = Vrai.
Les propriétés peuvent être lues et donc renseigner le programme sur l'opération à effectuer. Ainsi, si la feuille "seconde" n'est pas affichée, alors le programme l'affichera :

```
Sub laffichage()
    if Sheets("seconde").Visible = False then Sheets("seconde").Visible = True
End Sub
```

On peut ainsi construire une commande "basculante" inversant une propriété donnée :

```
Sub labascule()
    Sheets("seconde").Visible = Not (Sheets("seconde").Visible)
End Sub
```

a) Pour définir le contenu d'une cellule ou d'une plage de cellules

- Value Valeur numérique
- Formula Texte ordinaire
- FormulaR1C1 Formule de calcul
- Text Dans un graphique
- Name D'une cellule ou d'une feuille de calcul

b) Pour définir la forme d'une cellule ou d'une plage de cellules

Font (Police), FontStyle (Style de Police), Style (Style), FormatName (Format), NumberFormat (format de nombre).
Ces informations sont suivies d'un texte :

```
Selection.NumberFormat = "#.##0,00"
Selection.NumberFormat = "#.##0,00"" E"""
```

Note dans ce dernier format de nombre, le E pour Euros doit être entre guillemets. Dans la ligne de programme, le guillemet doit aussi être à l'intérieur des guillemets, par conséquent les guillemets sont doubles !

c) Propriétés définies par Vrai ou Faux (quelques exemples)

- DisplayStatusBar Affichage Barre Etat
- DisplayFormulaBar Affichage Barre Formule
- DisplayScrollBars Affichage Barres Défilement

- Visible Affiché (pour les objets ou les feuilles de calcul)
- ScreenUpdating Mise à jour de l'écran pendant une macro
- CutCopyMode Pour supprimer les encadrements autour des cellules copiées
- Saved Enregistré

d) Propriétés à définir en leur donnant une valeur (quelques exemples)

Pour la feuille : Zoom = 150
Pour les cellules : ColumnWidth = 5 (LargeurColonne)

Pour les graphiques : MaximumScale = 100 Echelle Maximum
Pour les dessins : Left et Top (Haut) pour positionner le dessin par exemple
Height et Width pour définir la taille des dessins

Pour la mise en page Header (EnTête), LeftHeader, RightFooter (Pied Page Droit)

Conseil : faire enregistrer les données plutôt que d'essayer de les écrire.

e) Propriétés désignant les conditions de travail

Selection, ActiveCell, ActiveWindow, ActiveWorkbook
Row, Column ou Address Indiquent la ligne, la colonne ou l'adresse complète de la cellule active
EntireColumn Ou EntireRow Pour étendre la sélection à la colonne ou à la ligne
Count Pour le dénombrement d'une collection

D) Opérateurs

Not, And, Or, Is, Like équivalent à Non, Et, Ou, Est, Comme

E) Instructions de VBA

Affectation d'une plage à des variables : Set puis nom de la variable et définition de la plage

- Set données1 = Evaluate ("L5") *Toute la ligne 5*
- Set données2 = Range("données") La plage doit être déjà nommée de manière unique dans le classeur
- Set données3 = [données] Les guillemets sont ici inutiles, idem ci-dessus
- Set données4 = Evaluate("L" & n_ligne & "C1:L" & autre_ligne & "C6")
- Set données5 = Evaluate ("[Nom_du_Classeur]Nom_de_lafeuille!nom_cellule")

Cette instruction permet d'utiliser ensuite le nom de la variable au lieu de l'adresse.

Ex : Données1(5).Value permet de lire la cinquième colonne de la ligne L5 c'est-à-dire L5C5.

Cette manière de procéder permet de simplifier (car les adresses n'ont pas à être répétées)
et d'accélérer le traitement d'une plage de cellules (car les cellules ne sont pas sélectionnées).

Exemple : transformer une plage de cellules contenant des valeurs en francs par des Euros :

```
Sub methode1() ' définition de chaque cellule. Il n'est pas nécessaire de les sélectionner
    For lacolonne = 3 To 6
        For laligne = 5 To 28
        Cells(laligne, lacolonne).Value = Cells(laligne, lacolonne).Value / 6.55957
      Next laligne
    Next lacolonne
End Sub

Sub methode2()
    Set Lescellules = Range("C5:F28")                    'nomme la plage
    nombre_de_cellules = lescellules.Count               'compte le nombre de cellules
    For numéro_cellule = 1 To nombre_de_cellules 'faire autant de fois que de cellules
        lescellules(numéro_cellule).Value = lescellules(numéro_cellule).Value / 6.55957
    Next numéro_cellule
End Sub

Sub methode3()                                           .

    Set lescellules = Range("C5:F28")
    For Each cellules In lescellules                 'la fonction Each évite de compter
        cellules.Value = cellules.Value / 6.55957        'la macro se charge
    Next                                                 'seule de traiter toutes les cellules
End Sub
```

```
Sub methode4()                                          'ne traite que les cellules selectionnées
    For Each cellules In Selection                      'le danger est de traiter des cellules non
        cellules.Value = cellules.Value / 6.55957       'connues
    Next
End Sub
```

F) Fonctions

- Fonctions numériques : Fix (Partie Entière), Int (Valeur Entière), Abs, Rnd (Valeur Aléatoire)
- Fonction texte : Chr (caractère), Len (nombre de caractères), Ucase (Majuscule), Lcase (Minuscule) Left, Right, Mid pour découper un texte, Str pour transformer une chaîne en nombre
- Fonctions date : Year, Month, Day, Date (date du système), DateSerial (construction d'une date),
- Fonctions heure : Hour, Minute, Second, Time (Heure du système), TimeSerial (construction d'une date),
- Fonctions de dialogue avec l'utilisateur : Msgbox, Inputbox
- Fonction tableau : Array (renvoie une variable tableau) qui débute à Base. (Cf. chapitre 4)

```
Sub letableaudesjours()
    lesjours = Array("lundi", "mardi", "mercredi", "jeudi", "vendredi", "samedi", "dimanche")
    For i = 1 To 7
        MsgBox lesjours(i)
    Next
End Sub
```

Sans autres précisions, la lecture commence le … mardi. En effet, l'initialisation par défaut est à 0. Pour lire tous les jours, il faudrait que la boucle aille de 0 à 6. Pour modifier l'initialisation des tableaux, il faut écrire dans la zone des déclarations

```
Option Base 1
```

- Les fonctions spécifiques d'Excel peuvent être appelées en les faisant précéder par Application.

```
Application.Count(laplage)          'pour utiliser la fonction NB d'Excel
```

- Gestion des répertoires :

CurDir *(répertoire courant),*
ChDir *(pour le modifier),*
MkDir *(en créer un nouveau),*
RmDir *(pour le renommer),*
Dir *(pour lire le contenu d'un répertoire),*
ChDrive *(pour changer de lecteur).*

Toutes ces fonctions sont suivies du nom du lecteur ou du répertoire comme texte (donc entre guillemets) par exemple:

```
Chdir "C:\victor"
```

Chapitre 4 : Techniques avancées de programmation

A) La déclaration des variables

VBA ne demande pas de déclaration systématique des variables.
Il peut cependant être intéressant de le faire pour indiquer ce que doit contenir la variable.
Les variables tableaux doivent absolument être déclarées.

a) Dim Variable as …

Déclarer une variable permet de prévenir le langage de l'espace nécessaire pour stocker les valeurs. Les compteurs sont généralement des entiers. En les déclarant comme tel, la mémoire est moins chargée.
Les principaux types sont :

Boolean : 0 ou 1, True ou False
Integer : Entier
String : Chaîne de caractères
Object : Défini par le programmeur
Variant : indéterminé (définition par défaut)

```
Sub En_entier()
    Dim i As Integer
    i = 1.4
    MsgBox i
End Sub
```

Ce programme n'est pas cohérent, la valeur renvoyée est arrondie à l'entier le plus proche

Les variables peuvent être déclarées au niveau de la procédure (supra) ou du module, dans la zone de déclaration avant la première procédure.

b) Public Variable as …

L'instruction Public, utilisée en remplacement de Dim au niveau du module rend accessible la variable à toutes les procédures de tous les modules durant le déroulement d'une procédure. La valeur est conservée tant que le classeur n'a pas été fermé.

Le nom du fichier courant lu dans la première procédure est utilisé dans le sous-programme sans autre transfert contrairement à l'autre version.

```
'programmme utilisant la déclaration

Public lefichier as String

Sub Publicité()
    lefichier = ActiveWorkbook.Name
    suite
End Sub

Sub suite()
    Msgbox lefichier
End Sub
```

```
'Autre procédure sans utilisation de la
'déclaration des variables

Sub Publicité()
    lefichier = ActiveWorkbook.Name
    suite lefichier
End Sub

Sub suite(lefichier)
    Msgbox lefichier
End Sub
```

c) Private Variable as …

Permet de déclarer des variables privées dans un module. Elles ne sont donc pas accessibles aux autres modules. D'où l'intérêt de créer des modules cohérents regroupant des procédures d'une même famille.

d) Static Variable as …

Officialise le souhait du programmeur de conserver la valeur donnée à la variable. Même si la déclaration Public peut donner les mêmes résultats, cette déclaration est préférable. La valeur est conservée tant que le classeur n'a pas été fermé. A la réouverture, les valeurs sont réinitialisées même si elles sont déclarées Static.

```
Sub conservation_valeur()
    Static lavaleur
    lavaleur = 1 + lavaleur * 2
    MsgBox lavaleur
End Sub
```

A chaque exécution de cette macro, la valeur affichée augmente 1, 3, 7, 15, etc.…

B) Les tableaux

Un tableau est une variable qui contient plusieurs valeurs. Chaque valeur est repérée par ses indices.
Un vecteur est un tableau simple à une dimension : Ensemble(10) donne la dixième valeur d'un vecteur
Le tableau peut avoir plusieurs dimensions Ensemble (10,3), mais la taille mémoire occupée s'accroît rapidement et les programmes deviennent difficiles à comprendre.
Une variable tableau doit être déclarée au niveau du module ou de la procédure.

La déclaration dans la procédure l'emporte sur toutes les autres.
Sauf à utiliser l'option Base les indices des valeurs commencent à 0. Cette option est valable pour tous les tableaux et doit être placée en tête du module.

L'exemple ci-dessous lit les trente valeurs d'une plage nommée Résultats et enregistre dans un tableau VBA les seules valeurs positives. Il annule les autres. Dans la seconde partie, il recopie les valeurs dans une nouvelle plage (juste pour la démonstration d'une procédure inverse !).

```
Option Base 1
Dim ensemble(10, 3) As Variant
Sub letableau()
    set données = [Résultats]
    For i = 1 to 10
        For j = 1 to 3
            ensemble(i, j) = Application.Max(0, données(i, j).Value)
        Next
    Next

    set Données2 = [Bis]
    For i = 1 to 10
        For j = 1 to 3
            Données2(i, j).Value = ensemble(i, j)
        Next
     Next
End Sub
```

La fonction Array permet de créer un tableau de constantes :

```
LesCodes = Array("CH";"CB";"PL";"LIQ")
```

Les tableaux s'utilisent de la même façon quel que soit le mode de leur création.

```
Sub Tableau_simple_0()
    LesCodes = Array("CH", "CB", "PL", "LIQ")
    Msgbox LesCodes(1) ' renvoie CB
End Sub
```

```
Option DébutTableau 1
Sub Tableau_simple_1()
    LesCodes = Array ("CH", "CB", "PL", "LIQ")
    Msgbox LesCodes(1) ' renvoie CH
End Sub
```

Une fois déterminée la taille d'un tableau, la procédure doit respecter la taille fixée sous peine d'interruption du programme par un message d'erreur ("indice en dehors de la plage").

Le tableur est un… tableau. Il n'est donc pas souvent nécessaire de créer des variables tableaux pour effectuer des calculs. Par contre une telle opération peut être utilisée pour traiter des données sans abîmer les données initiales et sans utiliser de feuilles de calcul.

Exemple : lecture des fichiers de type xls d'un répertoire

```
Option Base 1
Dim ensemble(100)                    ' Déclaration surdimensionnée volontairement
Sub letableau()
    lefichier = Dir("*.xls")   ' Cette fonction lit le premier fichier du type indiqué
    While NbrCar(lefichier) > 0       ' Pour arrêter la recherche lorsqu'il n'y a plus de fichiers
        i = i + 1
        ensemble(i) = lefichier
        lefichier = Dir()       ' Cette fonction lit le prochain fichier du même type
    Wend
End Sub
```

Macro permettant de lire un des trois cours de la semaine boursière pour l'une des 256 valeurs du règlement mensuel selon la demande de l'utilisateur. Le code et le nom de la valeur sont stockés systématiquement.
Le tableau initial se présente de la façon suivante :

Code	Titre	Date	N1	Ph	Pb	Der
3153	ROYAL CANIN	05/02/99	43	45,88	43	45,4
3153	ROYAL CANIN	08/02/99	46,5	46,65	43,6	44,5
3153	ROYAL CANIN	09/02/99	45,5	45,7	43,5	44
3153	ROYAL CANIN	10/02/99	44	44,45	43,5	43,5
3153	ROYAL CANIN	11/02/99	44	45,3	43,7	45
3153	ROYAL CANIN	12/02/99	46,4	46,4	45,65	45,85
3340	FONCIERE	05/02/99	130,1	131,8	130,1	131,8
3340	FONCIERE	08/02/99	130,5	131,5	128	130
…..	….	…..	….	….	….	…..

```
Option Base 1
Dim cours(256, 6, 7)

Sub remplissage()
   Set lescoursdo = [lescours]
   For i = 1 To 256
      For j = 1 To 6
         For k = 1 To 7
            cours(i, j, k) = lescoursdo((i - 1) * 6 + j, k)
         Next
      Next
   Next
   réponse = vbYes
   While réponse <> vbNo
      x1 = Str(InputBox("Titre", "Recherche", 2, 1, 1))
      x2 = Str(InputBox("Quelle date", "Recherche", 2, 1, 1))
      x3 = Str(InputBox("Quel type de cours", "Recherche", 2, 1, 1))+3
      réponse = MsgBox(cours(x1, x2, x3), vbYesNo)
   Wend

End Sub
```

C) Appel de procédures dans des procédures.

Il est souvent conseillé de décomposer une procédure en sous-procédures pour clarifier le programme ou simplement pour éviter des répétitions de lignes de programme.
Pour appeler une autre procédure du même projet, il suffit de donner son nom. Dans l'exemple 1 ci-dessous, la procédure principale appelle la procédure Sous_programme et l'exécute puis revient dans la macro principale.
Une procédure peut passer des informations à une sous-procédure (Exemple 2).

Exemple 1

```
Sub Principale()
      Sous_programme        ' le nom est libre
      MsgBox "le second message"
End Sub

Sub Sous_programme()
      MsgBox "le premier message"
End Sub
```

Exemple 2

```
Sub Principale1()
    Sous_programme2 "le premier message"
    Sous_programme2 "le second message"
End Sub

Sub Sous_programme2(texte_transféré)
    MsgBox texte_transféré
End Sub
```

Comble de raffinement, une procédure peut s'appeler elle-même (ou récursivité). Mais il faut prévoir un arrêt. Sinon, elles s'empilent les unes sur les autres et génèrent un message d'erreur. A manier avec précaution !

L'exemple ci-dessous a pour objectif de tirer un nombre au hasard (Rnd) entre 0 et 100 jusqu'à ce que soit obtenue une valeur comprise entre 5 et 10 ou entre 90 et 95 (toutes bornes exclues).

```
Sub récurrence()
    hasard = Rnd * 100
    If (hasard > 5 And hasard < 10) Or (hasard > 90 And hasard < 95) Then
        MsgBox "valeur finale " & hasard
        Exit Sub
    End if
    MsgBox hasard 'pour vérifier qu'il y a bien d'autres chiffres tirés
    récurrence
End Sub
```

D) La gestion des erreurs

Un programme ne devrait pas comprendre d'erreurs… mais elles ne peuvent être toujours éliminées.
Par exemple, une recherche dans une feuille n'est pas toujours couronnée de succès. Or il est possible que le programme cherche si la valeur existe ou non. Si la fonction Find ne trouve rien, elle génère une erreur. Il faut donc réagir à cette absence.

Les deux solutions principales sont les suivantes :
On Error Resume Next ignore l'erreur et poursuit le programme à la prochaine instruction. Dangereux car les erreurs ne sont plus signalées par le programme.

On Error GoTo line branche le programme sur le numéro de ligne ou l'étiquette (texte suivi de deux points :) en cas d'erreur d'exécution. La ligne indiquée doit se trouver dans la même procédure que l'instruction On Error. Une seule gestion d'erreur est possible dans une procédure. S'il y a plusieurs contrôles à effectuer, il faut alors utiliser des sous programmes.

```
Sub Gestion_contrôlée()
On Error GoTo lasuite
   Cells.Find("DECAN").Activate                 'Ce mot n'existe pas dans la feuille
   Exit Sub                                     'nécessaire pour éviter le message
lasuite:                                        'ligne de branchement du contrôle d'erreur
   MsgBox "Cette recherche n'a rien donné"
End Sub
```

```
Sub Dédain_derreur()
    On Error Resume Next                        'l'erreur est ici dédaignée
    Cells.Find("DECAN").Activate
    MsgBox "Le programme continue"
End Sub
```

E) Les macros automatiques

La plupart des macros sont lancées par l'utilisateur en cliquant sur un bouton de la barre d'outils, un objet dessiné ou un article de menus. Il est également possible que des macros se déclenchent toutes seules lors d'un événement répertorié : ouverture d'un classeur, changement de feuille de calcul, entrée d'une information etc.

Dans l'interface VBA ces macros automatiques sont très faciles à créer : il suffit de se porter sur l'objet concerné (par exemple la feuille), puis de choisir l'événement qui déclenchera la macro dans la boite de dialogue se trouvant en haut à droite. Ensuite il appartient au programmeur de gérer les opérations comme dans une macro normale. Les procédures sont toujours Private (privée) dans la mesure où elles ne concernent que l'objet précis pour lequel elles ont été créées. Toutes les opérations ne sont donc pas possibles

a) Procédures automatiques pour les feuilles de calculs

Target est la cellule la plus proche ou la dernière modifiée avant le déclenchement de la macro.
En activant la feuille

Procédure qui actualise automatiquement un tableau croisé dynamique lorsque la feuille qui le contient est sélectionnée.

```
Private Sub Worksheet_Activate()
 ActiveSheet.PivotTables("Tableau croisé dynamique1").RefreshTable
End Sub
```

En quittant la feuille

```
Private Sub Worksheet_Deactivate()
```

Avant le clic droit de la souris

```
Private Sub Worksheet_BeforeRightClick(ByVal Target As Excel.Range, Cancel As Boolean)
```

Avant le double clic de la souris

```
Private Sub Worksheet_BeforeDoubleClick(ByVal Target As Excel.Range, Cancel As Boolean)
```

Après une modification de la feuille de calculs

```
Private Sub Worksheet_Change(ByVal Target As Excel.Range)
```

Lorsque la sélection est modifiée

```
Private Sub Worksheet_SelectionChange(ByVal Target As Excel.Range)
```

Lorsque le recalcul est lancé

```
Private Sub Worksheet_Calculate()
```

b) Procédures automatiques pour les Classeurs

Toujours Private Sub. Se porter sur l'objet ThisWorkbook, puis choisir Workbook a lieu de Général. Choisir l'événement déclencheur de la macro. Par ex :

```
Workbook_Open()          ' en ouvrant le classeur
Workbook_BeforeClose     ' en fermant le classeur
Workbook_BeforePrint     ' avant l'impression
Workbook_BeforeSave      ' avant la sauvegarde
Workbook_NewSheet                ' avant la création d'une nouvelle feuille.
```

F) Description des macros

Les noms des macros ne sont pas toujours aussi clairs que souhaités particulièrement pour un utilisateur qui n'a pas créé le programme.
Il est possible de donner une brève description des effets de la macro

Soit dès l'enregistrement automatique de la macro

Soit dans VBA, en choisissant : Explorateurs d'objet (F2), puis le projet concerné, puis la macro et lire ses propriétés (BDS). Ecrire ou modifier la description

Lors d'une affectation d'une macro à un bouton ou un élément de menu, la description de la macro apparaît en bas de la boite de dialogue.

G) Exemples de macro utiles

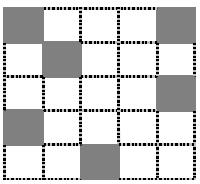
a) Repérer si la cellule active fait partie d'une zone déterminée :

```
Sub intersection()
  If Application.Intersect(ActiveCell, Range("B2:C10000")) Is Nothing Then
                    'cette procédure inverse permet d'éviter un message d'erreur
     MsgBox "pas d'intersection"
  Else
     MsgBox " intersection"
 End If
End Sub
```

L'adresse de comparaison peut correspondre à une plage nommée dans la feuille

```
… Intersect(ActiveCell, [lazonechoisie])…
```

Cette zone choisie n'est pas nécessairement rectangulaire, elle relève des définitions générales des noms dans Excel. Le déclenchement de la macro peut donc très bien correspondre à des sélections discontinues comme les cellules grisées du tableau ci-contre.



P.S. Les remarques, les corrections peuvent être adressées par mail à : jvictor@u-paris10.fr