Visual Basic pour Excel

Tableurs et comptes nationaux

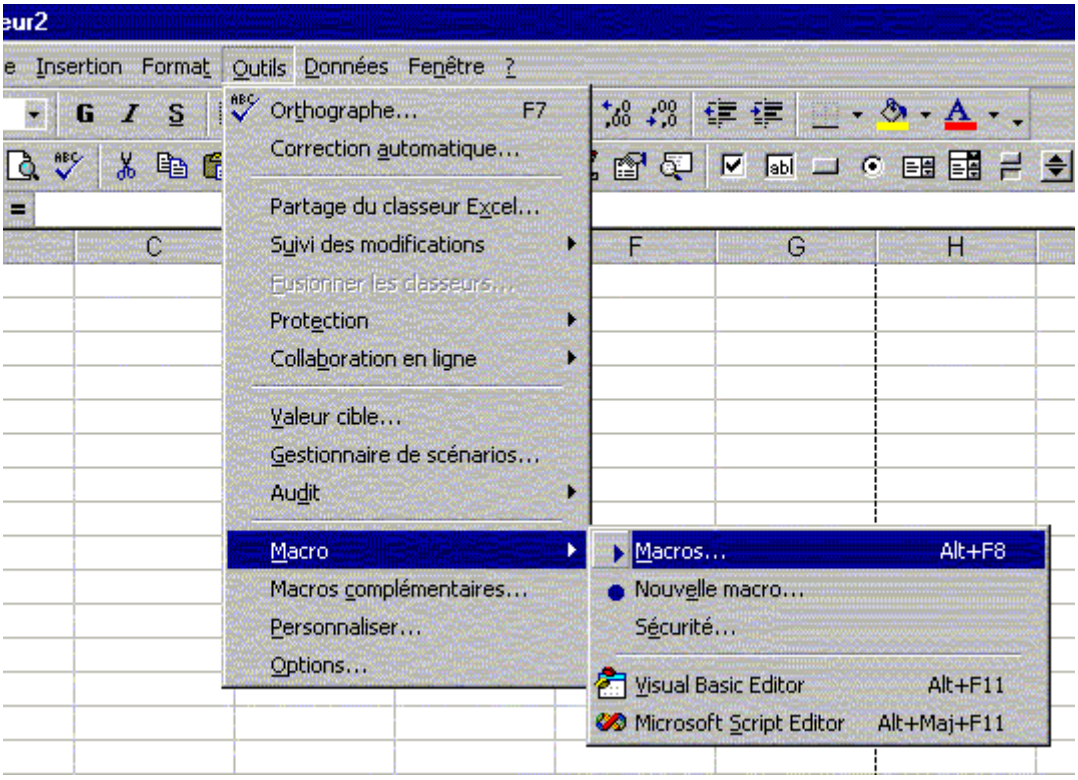
Les comptables nationaux utilisent généralement des logiciels développés spécialement à leur attention pour répondre à leurs besoins spécifiques. Cependant, il est rare que tout le travail d'élaboration des comptes nationaux puisse être réalisé intégralement avec ces logiciels. En effet, les comptables nationaux doivent parfois s'adapter à des circonstances exceptionnelles, par exemple lorsque de nouvelles sources apparaissent ou que des anciennes disparaissent de manière imprévue. De plus, les experts en charge de la vérification des comptes nationaux des états membres de l'Union Européenne ne peuvent généralement pas disposer de tels logiciels et ils doivent être capables de s'adapter à une grande variété de situations.

Les tableurs, notamment Excel qui le plus utilisé actuellement, peuvent s'avérer extrêmement utiles pour eux, pour ne pas dire indispensables. Cependant, l'utilisation d'un tableur pour traiter de grandes masses de données n'est pas si simple et les erreurs ne sont pas si faciles à détecter lorsque de nombreux calculs doivent s'enchaîner. Aussi, est-il important d'utiliser les tableurs de la manière la plus efficace possible en utilisant leurs capacités les mieux adaptées au travail d'élaboration des comptes nationaux. Parmi ces capacités figure incontestablement la disponibilité d'un langage de programmation. Pour Excel ce langage de programmation est Visual Basic.

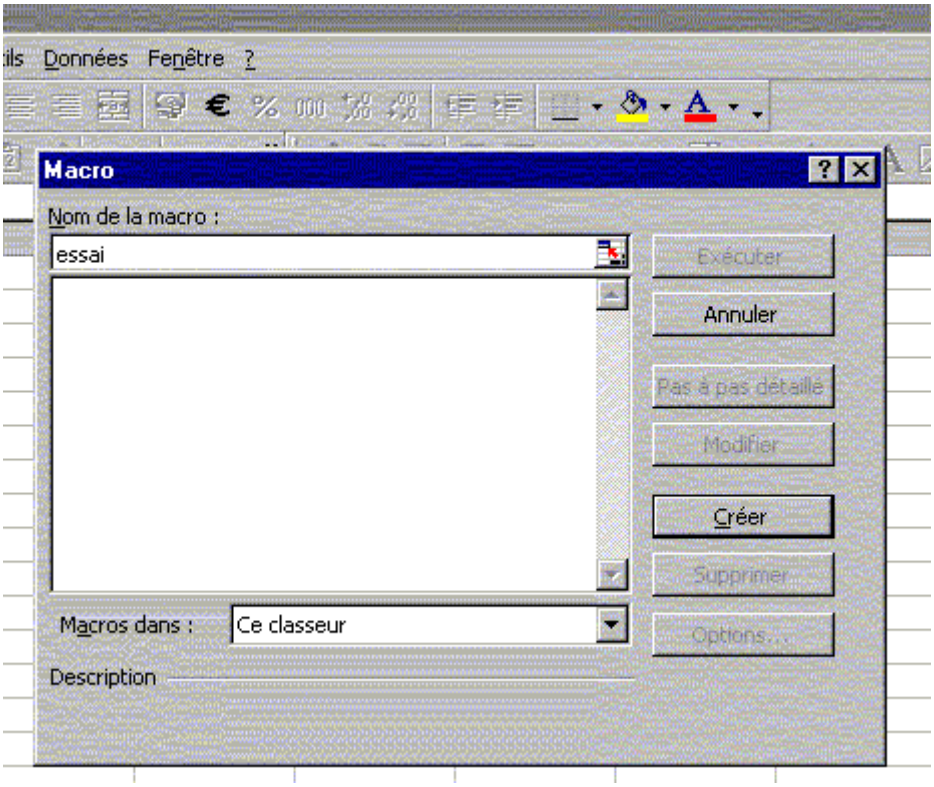
Création d'une macro

Visual Basic est un langage de programmation qui confère à Excel une grande puissance. Il peut être utilisé pour traiter de nombreux problèmes, par exemple le passage des données des comptabilités des entreprises aux estimations de la comptabilité nationale. Nous allons donc présenter quelques-unes de ses fonctionnalités dans les rubriques suivantes :

Le langage Visual Basic étant utilisé dans les macros d'Excel nous devons d'abord créer une macro. Ouvrons donc une nouvelle feuille d'Excel et allons dans le menu *Outils* de la barre de menus. Sélectionnons l'option *Macro* puis, dans le menu déroulant *Macros*.



Dans la zone *Nom de la macro* qui apparaît alors écrivons le nom que nous avons sélectionné pour la macro, par exemple *Essai*, puis cliquons sur le bouton *Créer*.



On passe alors dans l'éditeur Visual Basic. Dans la zone située à droite de l'écran apparaissent les lignes suivantes :

```
Sub essai()

End Sub
```

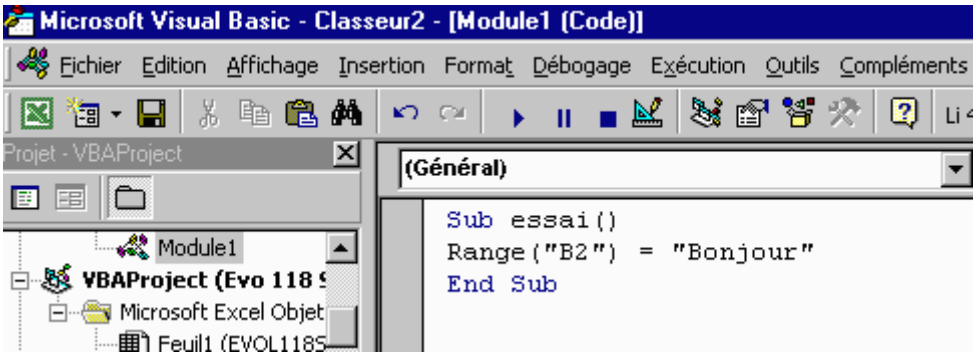
C'est entre les instructions Sub essai() et End Sub que devra être écrit notre programme Visual Basic.

Notre premier programme

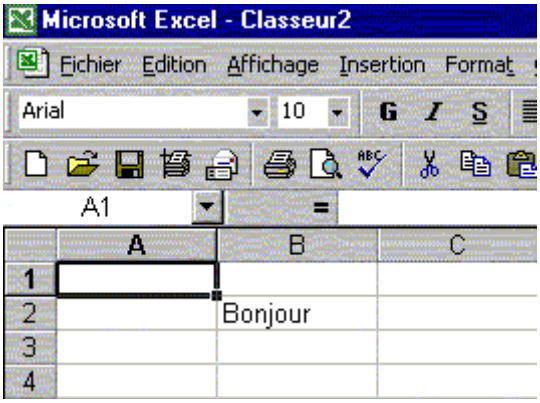
Notre premier exercice consistera à faire apparaître *Bonjour* dans la cellule B2. Pour cela nous allons taper l'instruction

```
Range("B2") = "Bonjour"
```

entre les instructions *Sub essai* et *End Sub*. Nous obtenons :



En cliquant à l'intérieur du programme puis sur la flèche  du menu nous lançons le programme et, miraculeusement (ou presque !), nous voyons apparaître *Bonjour* dans la cellule B2 de la feuille Excel.

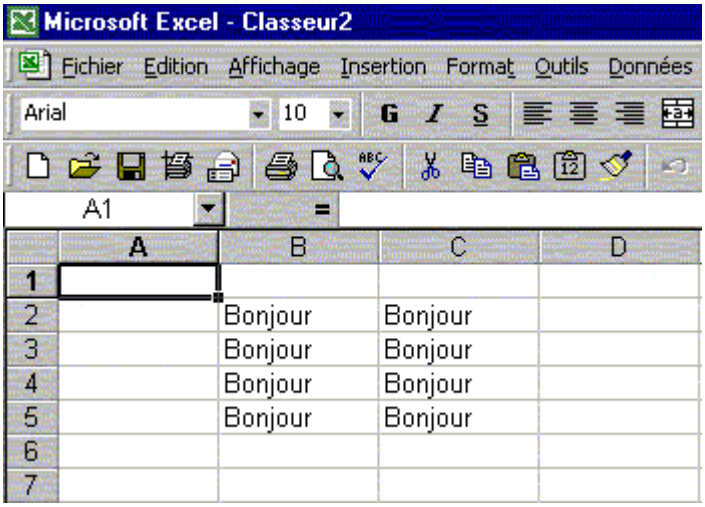


Nous aurions également pu lancer notre macro depuis le menu de la feuille Excel en sélectionnant dans le menu *Outils Macro* puis *Macros* et le nom de la macro.

Expliquons donc comment fonctionne ce programme. *Range* désigne un champ de la feuille en cours, c'est-à-dire un ensemble de cellules. Ici, le champ est composé d'une seule cellule et est repéré par ses coordonnées. Notons la présence obligatoire de guillemets, faute de quoi le programme comprendrait que B2 est une variable ayant la valeur zéro. Nous attribuons au champ la valeur *Bonjour*. Là encore la présence de guillemets est indispensable. Le champ peut aussi être un ensemble de cellules, par exemple si nous écrivons :

```
Sub essai()
Range("B2:C5") = "Bonjour"
End Sub
```

Nous obtenons :



Si nous attribuons un nom au champ dans la feuille Excel, nous pouvons dans la macro remplacer les coordonnées par le nom du champ, toujours placé entre guillemets.

Un classeur Excel étant composé de différentes feuilles, il est généralement intéressant de préciser dans quelle feuille on veut envoyer les données. En effet, par défaut, les données sont envoyées dans la feuille active mais, lorsqu'on lance la macro, la feuille active n'est toujours celle que nous voudrions modifier. Aussi, nous pouvons préciser la feuille de travail en la désignant soit par son numéro, soit par son nom. Par exemple, si nous souhaitons écrire *Bonjour* sur la deuxième feuille nous pouvons écrire soit :

```
Sub essai()
Sheets(2).Range("B2:C5") = "Bonjour"
End Sub
```

Soit :

```
Sub essai()
Sheets("Feuil2").Range("B2:C5") = "Bonjour"
End Sub
```

Sheets représente l'ensemble des feuilles d'un classeur, Sheets(2) représente la deuxième feuille du classeur actif. Visual Basic est un langage orienté objet et l'on voit ici apparaître la logique de sa syntaxe. Sheets est une collection d'objet, Sheets(2) un objet qui possède un champ. La description va du général au particulier, de l'objet à ses composantes, chaque élément étant séparé par un point.

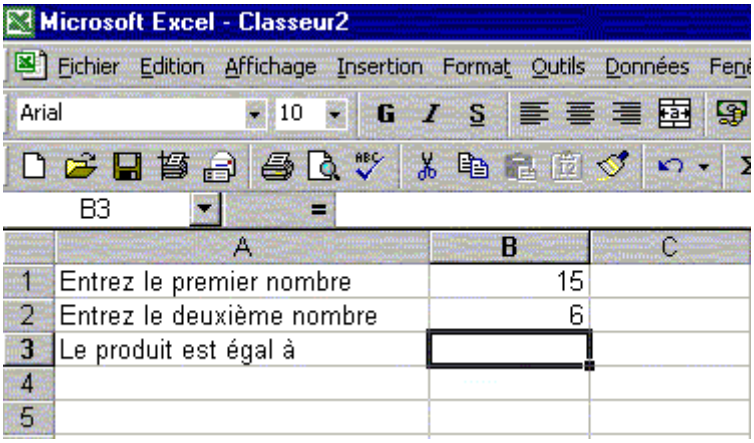
Nous aurions également pu préciser le nom du classeur. Dans notre exemple il s'appelle *Classeur2* et nous pouvons donc écrire :

```
Sub essai()
Workbooks("Classeur2").Sheets("Feuil2").Range("B2:C5") = "Bonjour" End Sub
```

Workbooks désigne l'ensemble des classeurs et Workbooks("Classeur2") le classeur qui a pour nom *Classeur2*.

Utiliser des variables

Visual Basic n'est utile que dans la mesure où il peut traiter des informations que lui communique l'utilisateur. Pour cela il utilise des variables où il va stocker les données. Nous allons prendre l'exemple d'une macro qui permet de calculer le produit de deux nombres. Supposons donc que nous ayons créé une feuille se nommant *Produit* et se présentant de la manière suivante :



Nous allons stocker le premier nombre dans une variable que nous appellerons *a* et le deuxième dans une variable que nous appellerons *b*. Le résultat sera stocké dans une variable *c*. La macro *Calcul* fera le calcul et fera apparaître 90 dans la cellule *B3* :

```
Sub Calcul()
Set f = Sheets("Produit")
a = f.Range("B1")
b = f.Range("B2")
c = a * b
f.Range("B3") = c
End Sub
```

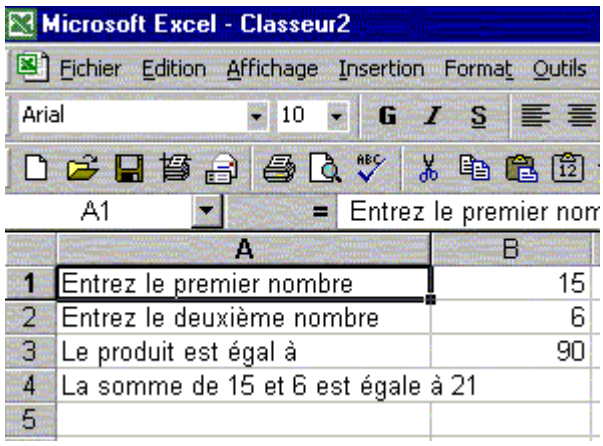
Dans cette macro nous avons d'abord introduit une variable *f*. C'est une variable dite variable objet car elle ne représente pas une valeur mais un objet, qui est ici la feuille *Calcul*. Cette variable doit être introduite par l'instruction *Set*. Nous introduisons ensuite les deux variables *a* et *b* auxquelles nous attribuons les valeurs des cellules *B1* et *B2*. Ces deux variables ne doivent pas être introduites par l'instruction *Set* car elles sont destinées à contenir des valeurs. Remarquons que *f.Range("B1")* représente le champ *B1* de la feuille *f*, c'est-à-dire de la feuille *Produit*. Le produit est ensuite stocké dans la variable *c* et affiché dans la cellule *B3*. Notons ici que l'instruction c=a*b n'est pas une équation, elle signifie simplement que le contenu de la partie droite du signe égal est affecté à la variable située à gauche du signe égal, mais on ne pourrait pas écrire, par exemple, c+1=a*b

Les variables peuvent également contenir des textes, par exemple le programme suivant

```
Sub Calcul()
Set f = Sheets("Produit")
a = f.Range("B1")
b = f.Range("B2")
c = a * b
f.Range("B3") = c
d = "La somme de " & a & " et " & b & " est égale à " & a + b
f.Range("A4") = d
```

```
End Sub
```

donnera :



Dans cette macro & est l'opérateur de concaténation, il permet de combiner des textes et même des textes et des nombres car ceux-ci sont alors convertis en texte.

Travailler avec des boucles

L'une des forces principales des programmes et donc des macros est de permettre de réaliser très simplement des calculs itératifs. Ils utilisent pour cela des boucles. Supposons que nous cherchions à faire la somme des dix nombres de la feuille *Somme* suivante :

	A	B
1	N1	50
2	N2	100
3	N3	80
4	N4	75
5	N5	24
6	N6	32
7	N7	47
8	N8	52
9	N9	14
10	N10	75
11		
12	Somme	
13		

Le programme suivant :

```
Sub Somme()
Set f = Sheets("Somme")
t = 0
For i = 1 To 10
   a = f.Cells(i, 2)
   t = t + a
Next i
f.Cells(12, 2) = t
End Sub
```

fait apparaître la somme 549 en cellule *B12*

Dans cette macro, la variable t qui est destinée à recevoir la somme est d'abord initialisée à zéro, ce qui n'est pas obligatoire dans ce cas précis puisqu'une nouvelle variable est initialisée à zéro, mais c'est une règle de précaution lorsque l'on procède par itérations. La boucle correspond à l'instruction For ... Next. Cette instruction fait appel à une variable de comptage, ici *i* qui va prendre successivement toutes les valeurs comprises entre 1 et 10. Toutes les instructions comprises entre For et Next seront exécutées successivement pour chaque valeur de *i*, c'est-à-dire 1, 2, 3 ... 9, 10.

f.Cells(i,2) désigne la cellule située à la ième ligne de la deuxième colonne de la feuille f, c'est-à-dire la feuille *Somme*. Au début de la boucle, le compteur *i* prend la valeur 1. f.Cells(i,2) devient f.Cells(1,2) qui a la valeur 50, cette valeur est affectée à la variable *a*.

A la ligne suivante, dans la partie droite de l'égalité, la variable *t* vaut 0 car c'est la valeur qui lui a été affectée au début du programme, t+a vaut donc 0+50=50. Cette valeur de 50 est affectée à la variable située à gauche du signe égal, c'est-à-dire *t*. Le programme arrive ensuite à l'instruction Next i, il donne alors à *i* la valeur 2 et revient à la première ligne après l'instruction For.

La variable *a* prend alors la valeur de f.Cells(2,2), c'est-à-dire 100. A la ligne suivante, dans la partie droite, la variable *t* a la valeur 50 qui lui a été affectée au tour précédent, t+a a donc la valeur 50+100=150. Cette valeur est alors affectée à la variable de la partie gauche, c'est-à-dire *t* qui devient donc égale à 150. Le programme arrive ensuite à l'instruction Next et donne au compteur *i* la valeur 2. Il reprend ensuite à la première ligne située après l'instruction Next.

Ce processus va se poursuivre jusqu'à ce que le compteur parvienne à 10. A la fin de la dixième boucle le programme continue au delà de l'instruction Next. La variable *t* contient alors la somme des 10 nombres de la deuxième colonne de la feuille *Somme*. Cette somme est affichée dans la cellule correspondant à la douzième ligne de la deuxième colonne de la feuille *Somme*. Ce programme montre la manière classique de calculer une somme de valeurs avec Visual Basic mais il en existe d'autres.

Une autre manière de réaliser des boucles est d'utiliser l'instruction Do While...Loop. Celle-ci va effectuer une boucle tant qu'une condition est vérifiée. Par exemple, nous aurions pu écrire le programme précédent de la manière suivante :

```
Sub Somme1()
Set f = Sheets("Somme")
t = 0
i = 1
Do While i <= 10
   a = f.Cells(i, 2)
   t = t + a
   i = i + 1
Loop
f.Cells(12, 2) = t
End Sub
```

Dans cette macro la boucle va de l'instruction Do While à l'instruction Loop et elle est éffectuée tant que la variable *i* est inférieure ou égale à 10.

Il est possible également d'utiliser la boucle For each ... Next en association avec la fonction Array() afin de sélectionner certaines valeurs dans une liste. Par exemple, si nous voulons faire le toatal des lignes 2, 7 et 9, nous pouvons utiliser la macro suivante :

```
Sub Somme1()
Set f = Sheets("Somme")
```

```
t = 0
i = 1
For Each i In Array(2, 7, 9)
   a = f.Cells(i, 2)
   t = t + a
   i = i + 1
Next i
f.Cells(12, 2) = t
End Sub
```

Nous pouvons également utiliser Array pour une liste de valeurs alphanumériques à condition d'insérer ces valeurs entre guillemets.

Les variables multidimensionnelles

Il est possible de travailler dans Visual Basic avec des variables à plusieurs dimensions. Par exemple, nous pouvons vouloir saisir le tableau ci-dessous dans une variable à deux dimensions.

	A	B	C
1	10	80	
2	20	70	
3	30	60	
4	40	50	
5	50	40	
6			
7			

Pour cela nous devons au préalable utiliser l'instruction Dim qui précise la dimension de la variable. Ainsi, si nous désignons par A la variable dans laquelle nous voulons saisir le tableau, la macro suivante montre comment saisir le tableau dans la variable A pour le recopier ensuite plus bas et transposé en le relisant à partir de A. Le tableau est supposé être écrit dans la feuille *Tableau*

```
Sub Multi()
Dim A(5, 2)
Set f = Sheets("Tableau")
For i = 1 To 5
   For j = 1 To 2
      A(i, j) = f.Cells(i, j)
   Next j
Next i
For i = 1 To 5
   For j = 1 To 2
      f.Cells(j + 6, i) = A(i, j)
   Next j
Next i
End Sub
```

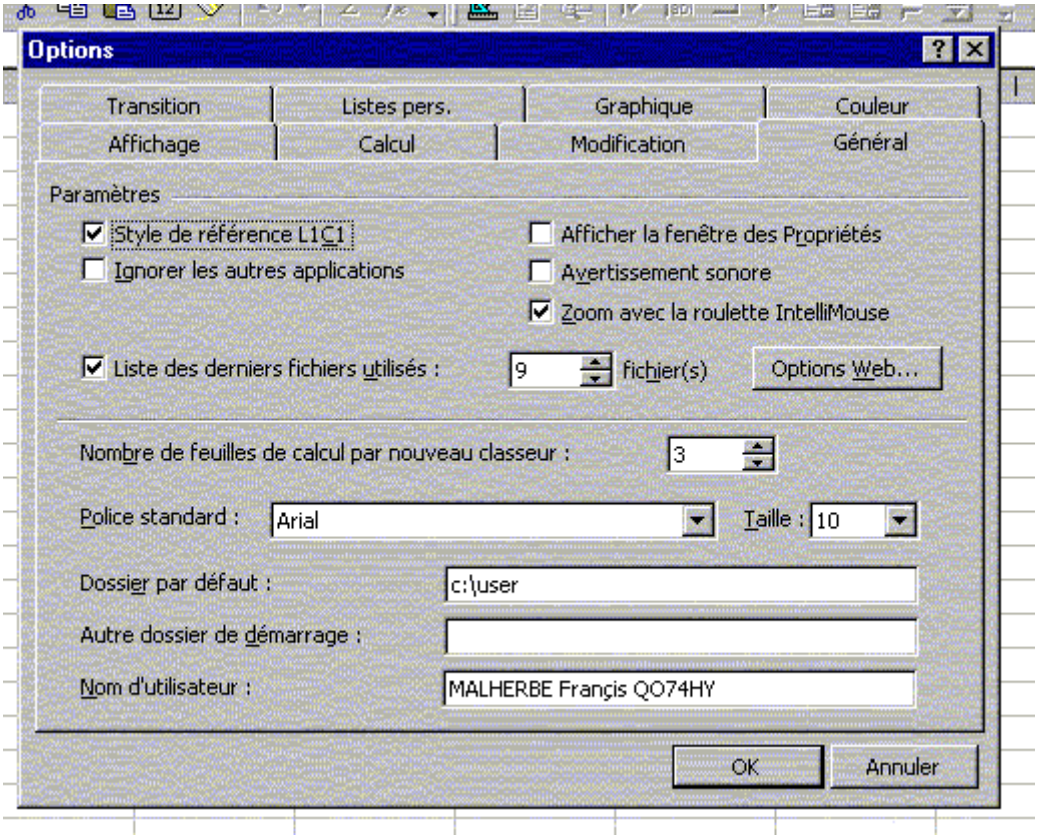
Le résultat sera le suivant :

	A	B	C	D	E
1	10	80			
2	20	70			
3	30	60			
4	40	50			
5	50	40			
6					
7	10	20	30	40	50
8	80	70	60	50	40
9					

L'instruction Dim précise que le tableau a 2 dimensions, la première repérée par un indice qui va de 0 à 5, la seconde par un indice qui va de 0 à 2. Ensuite, nous voyons l'utilisation de boucles imbriquées puisque nous avons deux groupes d'instructions For...Next, la première qui utilise le compteur *i*, la seconde qui utilise le compteur *j*. Ces deux boucles doivent être impérativement emboîtées, c'est-à-dire respecter l'ordre suivant :
For i...
...
For j...
...
Next j...
...
Next i...

Dans la première partie, la variable A est remplie à partir des cellules de la feuille, dans la seconde partie, les valeurs contenues dans la variable A sont recopiées sur la feuille en décalant les lignes de 6 vers le bas. La transposition se fait extrêmement simplement en intervertissant dans f.Cells(i,j) *i* par *j*.

REMARQUE : lorsqu'on utilise Cells(i,j), il peut être intéressant, au moins temporairement, de changer la présentation de la feuille de calcul en faisant apparaître les numéros des colonnes. Pour cela, il faut aller dans le menu *Outils* d'Excel, choisir *Options*, puis *Général* et cliquer sur l'option *Style de référence L1C1*.



Instructions conditionnelles

Il est souvent intéressant d'utiliser dans les macros des instructions qui ne seront exécutées que dans certaines circonstances. Cela peut être réalisé grâce à l'instruction IF...THEN. Par exemple, supposons que la feuille *Tableau* se présente comme ci-dessous et que l'on cherche à recopier dans la deuxième colonne tous les nombres supérieurs à 20.

	A	B
1	15	
2	50	
3	20	
4	30	
5	40	
6	55	
7		

La macro suivante obtient ce résultat :

```
Sub Test()
Set f = Sheets("Tableau")
For i = 1 To 6
   If f.Cells(i, 1) > 20 Then
      f.Cells(i, 2) = f.Cells(i, 1)
   End If
Next i
End Sub
```

Nous obtenons :

	A	B
1	15	
2	50	50
3	20	
4	30	30
5	40	40
6	55	55
7		

Cette macro nous montre la structure de l'instruction IF...THEN. La condition doit être placée après IF et avant THEN. Entre THEN et END IF doivent être placées les instructions qui ne seront exécutées que lorsque la condition sera remplie. Remarquons également comment procède la macro pour recopier une valeur d'une cellule dans une autre cellule, c'est la méthode la plus efficace pour le faire.

Nous pouvons introduire dans la condition des opérateurs comme AND ou OR. Par exemple, si nous voulons recopier dans la deuxième colonne uniquement les valeurs supérieures à 20 et inférieures à 50, nous pourrions utiliser la macro suivante :

```
Sub Test()
Set f = Sheets("Tableau")
For i = 1 To 6
   If f.Cells(i, 1) > 20 And f.Cells(i, 1) < 50 Then
      f.Cells(i, 2) = f.Cells(i, 1)
   End If
Next i
End Sub
```

Nous aurions obtenu :

	A	B
1	15	
2	50	
3	20	
4	30	30
5	40	40
6	55	
7		

Si nous avions voulu recopier les nombres inférieurs à 20 ou supérieurs à 30, nous aurions pu utiliser la macro suivante :

```
Sub Test()
Set f = Sheets("Tableau")
For i = 1 To 6
   If f.Cells(i, 1) < 20 Or f.Cells(i, 1) > 30 Then
      f.Cells(i, 2) = f.Cells(i, 1)
   End If
Next i
End Sub
```

Nous aurions obtenu :

	A	B
1	15	15
2	50	50
3	20	
4	30	
5	40	40
6	55	55
7		

Nous pouvons ajouter à l'intérieur de la séquence IF...THEN...END IF une clause ELSE afin de spécifier des instructions qui doivent être exécutées quand la condition n'est pas vérifiée. Par exemple, si nous voulons recopier dans la deuxième colonne les nombres supérieurs à 20 et recopier dans la troisième colonne les autres, nous pouvons utiliser la macro suivante :

```
Sub Test()
Set f = Sheets("Tableau")
For i = 1 To 6
   If f.Cells(i, 1) > 20 Then
      f.Cells(i, 2) = f.Cells(i, 1)
  Else
      f.Cells(i, 3) = f.Cells(i, 1)
   End If
Next i
End Sub
```

Nous obtenons :

	A	B	C	D
1	15		15	
2	50	50		
3	20		20	
4	30	30		
5	40	40		
6	55	55		
7				

Ce texte n'engage que son auteur : Francis Malherbe