

Introduction à Visual Basic pour Excel

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.

Pour utiliser le langage Visual Basic d'Excel, nous devons d'abord créer une macro. Ouvrons donc un classeur Excel et sélectionnons dans le menu le groupe *Développeur*. Si ce groupe n'apparaît pas dans le menu, nous devons aller dans *Fichiers* puis dans *Options*. Dans le cadre qui apparaît cliquons sur *Personnaliser le ruban*. Dans la partie droite du cadre, il faut cocher l'option *Développeur* puis cliquer sur *OK*.

Nous pouvons maintenant aller dans le menu et sélectionner le groupe *Développeur*. Pour créer une macro, il faut cliquer sur *Macros*. Apparaît alors le cadre *Macro* dans lequel nous devons rentrer le nom de la macro que nous voulons créer, puis cliquer sur *Créer*. Donnons donc à notre première macro le nom *essai*.

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

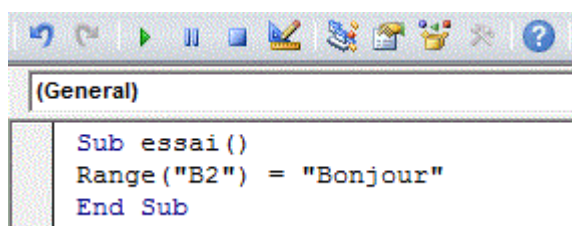
```
Sub essai()
```


```
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.

	A	B	C
1			
2		Bonjour	
3			

Nous aurions également pu lancer notre macro depuis le menu de la feuille Excel en allant dans le menu le groupe *Développeur* puis en cliquant sur *Macros*. Dans le cadre *Macro*, sélectionnons le nom de la macro et cliquons sur *Exécuter*, "Bonjour" apparaît alors.

Expliquons donc comment fonctionne ce programme. *Range* désigne une plage de cellules de la feuille en cours, c'est-à-dire un ensemble de cellules. Ici, la plage est composé d'une seule cellule et est repérée 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 à la cellule la valeur *Bonjour*. Là encore la présence de guillemets est indispensable. La plage peut aussi être un ensemble de cellules, par exemple si nous écrivons :

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

Nous obtenons :

	A	B	C	D
1				
2		Bonjour	Bonjour	
3		Bonjour	Bonjour	
4		Bonjour	Bonjour	
5		Bonjour	Bonjour	
6				

Si nous attribuons un nom à la plage dans la feuille Excel, nous pouvons dans la macro remplacer les coordonnées par le nom de la plage, toujours placé entre guillemets.

Des objets

En fait, une plage de cellules est ce que les informaticiens appellent un objet. Cet objet possède différentes propriétés que l'on peut modifier à partir de Visual Basic.

Quand nous avons écrit `Range("B2:C5") = "Bonjour"`, nous avons attribué une valeur à la plage de cellules. Nous aurions donc pu écrire plus précisément :
`Range("B2:C5").value = "Bonjour"`
Nous aurions obtenu le même résultat.

Nous pouvons changer d'autres propriétés de la plage de cellules, par exemple sa couleur :
`Range("B2:C5").Interior.ColorIndex = 6`
La plage de cellules apparaît alors avec un fond jaune.

Des collections

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
```

Pour Visual Basic, une feuille d'un classeur est un objet et l'ensemble des feuilles du classeur constitue une collection.

Sheets est la collection constituée de l'ensemble des feuilles d'un classeur, *Sheets(2)* est un objet de la collection *Sheets* qui représente la deuxième feuille du classeur actif.

Un langage orienté objet

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 des plages, les plages sont des objets possédant des caractéristiques.

Il y a une hiérarchie allant du classeur à la cellule. La description va du général au particulier, de la collection à l'objet, de l'objet à ses composantes, chaque élément étant séparé par un point.

Ainsi, 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").Value =  
"Bonjour"  
End Sub
```

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

Pour désigner le classeur contenant la macro, on peut écrire :
`ThisWorkbook.Sheets("Feuil2").Range("B2:C5").Value = "Bonjour"`

On peut également effacer le contenu d'une cellule ou d'une plage de cellules avec *ClearContents*. Par exemple, le programme ci-dessous efface le contenu de la plage "C2:C5" de la deuxième feuille :

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

Si on utilise `Clear` au lieu de `ClearContents`, on efface également le format.

Sélectionner des cellules

Un autre moyen de sélectionner une cellule est d'utiliser *Cells(i,j)*, où *i* désigne un numéro de ligne et *j* un numéro de colonne.

Par exemple, le programme suivant écrit *Bonjour* à l'intersection la troisième ligne et de la quatrième colonne de la feuille *Feuil1* :

```
Sub essai()  
ThisWorkBook.Sheets("Feuil1").Cells(3,4) = "Bonjour"  
End Sub
```

On peut appliquer *Cells* non seulement à une feuille mais aussi à une plage de cellules. Par exemple, le programme ci-dessous va écrire "Bonjour" dans la cellule "C4" :

```
Sub essai()  
Sheets("Feuil1").Range("B2:H8").Cells(3, 2) = "Bonjour"  
End Sub
```

En effet, la cellule "C4" est à l'intersection de la troisième ligne et de la deuxième colonne de la plage "B2:H8".

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 :

	A	B	C
1	Entrez le premier nombre	15	
2	Entrez le deuxième nombre	6	
3	Le produit est égal à		
4			

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 = Workbooks("Classeur2").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 *Produit*. 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 la plage *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 = Workbooks("Classeur2").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 :

	A	B	C
1	Entrez le premier nombre	15	
2	Entrez le deuxième nombre	6	
3	Le produit est égal à	90	
4	La somme de 15 et 6 est égale à 21		
5			

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.

Il est recommandé de définir les variables en début de programme grâce à l'instruction *Dim*. Cette instruction doit être suivie du nom de la variable puis de *As*, puis du type de la variable. Par exemple

```
Dim a As Integer
```

```
Dim c As Double
```

```
Dim d As String * 80
```

Pour définir une variable de type *String*, on peut définir sa longueur en la précédant d'une astérisque.

Les différents types de variables sont les suivants :

Boolean
Byte
Char (caractère unique)
Date

Decimal
Double (nombre à virgule flottante double précision)
Integer
Long (entier long)
object
SByte
Short (entier court)
Single (nombre à virgule flottante simple précision)
String (caractères, longueur variable)
UInteger
ULong
User-Defined (structure)
UShort

Incrémentation

Dans le langage Visual Basic, une variable fait référence à un emplacement de la mémoire de l'ordinateur. Lorsqu'on attribue une valeur à une variable en écrivant une égalité, le programme commence par évaluer l'expression qui se situe à droite du signe égal et place le résultat à l'emplacement défini par le nom de la variable qui est à gauche du signe égal. Par exemple, lorsqu'on écrit $c=a*b$, le programme va chercher le contenu des variables a et b , fait la multiplication et place le résultat dans l'emplacement correspondant à la variable c .

Il est donc possible d'écrire une même variable des deux côtés du signe égal. Cette possibilité est utilisée notamment dans le cas particulièrement important de l'incrémentatation. L'incrémentatation est une opération qui consiste à ajouter une valeur entière à un compteur. Par exemple, si i désigne une variable entière de valeur 10, on peut écrire :

$i = i + 1$

Après l'exécution de cette ligne la variable i sera égale à 11.

Les fonctions

Il existe un très grand nombre de fonctions disponibles dans Visual basic pour Excel. Par exemple, on peut utiliser les fonctions mathématiques suivantes :

Fonction	
Abs	Retourne la valeur absolue d'un nombre.
Cos	Retourne le cosinus de l'angle spécifié.
Exp	Retourne e (la base des logarithmes népériens) déclenché à la puissance spécifiée.
Log	Retourne le logarithme naturel d'un nombre spécifié ou le logarithme d'un nombre spécifié dans une base spécifiée.
Round	Retourne une valeur Decimal ou Double arrondie à la valeur intégrale la plus proche ou à un nombre de chiffres fractionnaires.
Sin	Retourne le sinus de l'angle spécifié.
Tan	Retourne la tangente de l'angle spécifié.

Par exemple, le programme suivant :

```
Sub Calcul()  
Set f = Workbooks("Classeur2").Sheets("Produit")  
f.Range("B1")=Abs(-34.54)  
f.Range("B2")=Cos(2.67)  
End Sub
```

Renvoie 34,54 dans la cellule "B1" et -0,890845867 dans la cellule "B2".

On peut également travailler avec les fonctions de la feuille de calcul grâce à *Application.WorksheetFunction*. Par exemple, le programme suivant renvoie 12,6 dans la cellule "B6" :

```
Sub Calcul()  
Set f = Workbooks("Classeur2").Sheets("Produit")  
f.Range("B6") = Application.WorksheetFunction.Max(8.5, 12.6)  
End Sub
```

On remarquera qu'en Visual Basic les nombres doivent être entrés avec le point décimal et que le séparateur est la virgule.

On peut également travailler avec des plages à condition de les définir comme des variables. Par exemple, le programme suivant renvoie la somme des cellules de la plage "B1:B4" dans la cellule "B5" :

```
Sub Calcul()
Set f = Workbooks("Classeur2").Sheets("Produit")
Set plage = f.Range("B1:B4")
f.Range("B5") = Application.WorksheetFunction.Sum(plage)
End Sub
```

Visual Basic fournit également des fonctions pour travailler avec des chaînes de caractère. Par exemple :

Fonction	Description
Chr	Retourne le caractère associé au code du caractère spécifié.
Format	Retourne une chaîne mise en forme conformément aux instructions contenues dans une expression String de format.
InStr	Retourne un entier spécifiant la position de début de la première occurrence d'une chaîne à l'intérieur d'une autre.
Left	Retourne une chaîne contenant un nombre spécifié de caractères en partant de la gauche d'une chaîne.
Len	Retourne un entier contenant le nombre de caractères dans une chaîne.
LTrim	Retourne une chaîne contenant une copie d'une chaîne spécifiée sans espaces à gauche.
Mid	Retourne une chaîne contenant un nombre spécifié de caractères d'une chaîne.
Replace	Retourne une chaîne dans laquelle une sous-chaîne spécifiée a été remplacée par une autre sous-chaîne, un nombre de fois déterminé.
Right	Retourne une chaîne contenant un nombre spécifié de caractères depuis la partie droite d'une chaîne.
RTrim	Retourne une chaîne contenant une copie d'une chaîne spécifiée sans espaces à droite.
Space	Retourne une chaîne composée d'un nombre spécifié d'espaces.
Trim	Retourne une chaîne contenant une copie d'une chaîne spécifiée sans espaces à gauche ni à droite.

Par exemple Replace("abcd", b, c) renvoie "accd".

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 = ThisWorkbook.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 *i*ème 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 effectuée 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 total 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 voyons 161 s'afficher dans la ligne 12.

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 copiées 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, dans le menu

d'Excel, il faut aller au groupe *Fichiers*, choisir *Options*, puis *Formules* et cocher l'option *Style de référence L1C1*.

Dimensions d'un tableau

Un tableau dont les dimensions sont définies au niveau de sa déclaration est dit de taille fixe. Il existe également des tableaux de taille variable, on les appelle tableaux dynamiques.

La dimension d'un tableau dynamique doit être définie par l'instruction ReDim. Par exemple :

```
Dim essai() As Integer
ReDim essai(5)
essai(5) = 10
```

La taille du tableau peut être redéfinie au cours du programme par l'instruction ReDim.

Un tableau peut également être défini comme étant du type Variant, ses dimensions seront alors définies par un tableau ou une matrice qu'on lui affecte.

Par exemple :

```
Dim B as Variant
B = Array(2, 6, 9, 23, 32)
```

Si on lui affecte ensuite un tableau de taille différente, sa dimension s'ajustera automatiquement.

Quand on ne connaît pas la dimension d'un tableau, par exemple parce qu'on l'a défini comme Variant, on peut utiliser les fonctions LBound et UBound. La fonction LBound permet de déterminer les indices les plus bas et la fonction UBound les indices les plus hauts.

Pour utiliser les fonctions LBound et UBound, il faut leur indiquer le nom du tableau et la dimension dont on veut connaître l'indice inférieur ou supérieur. Par exemple :

```
Dim A(1 To 5, 1 To 2)
UBound(A, 1)=5
UBound(A, 2)=2
```

Lorsqu'un tableau n'a qu'une dimension, il suffit de préciser son nom. Par exemple :

```
UBound(B)=4
```

Les indices du tableau B vont, en effet, de 0 à 4.

Par défaut, les indices d'un tableau commencent à zéro. Si l'on veut que tous les tableaux commencent avec l'indice 1, il faut indiquer *Option Base 1* au tout début du module où se trouve le programme, c'est-à-dire avant tous les programmes.

Si nous voulons qu'un seul tableau commence à l'indice 1, nous pouvons le préciser au niveau de la déclaration Dim, par exemple :

```
Dim A(1 To 5, 1 To 2)
```

Les tableaux imbriqués

Les cellules d'un tableau peuvent être elles-mêmes des tableaux. Par exemple :

```
Sub MacroTableau() Dim Tableau(1 To 3)
a = Array("Madame", "Monsieur")
b = Array(53, 23, 76, 18)
c = Array(1.567, 2.42, 3.57, 4.31, 5.1)
Tableau(1) = a
Tableau(2) = b
Tableau(3) = c
For col = LBound(a) To UBound(a)
    ThisWorkbook.Sheets(1).Cells(1, col + 1) = Tableau(1)(col)
Next col
For col = LBound(b) To UBound(b)
    ThisWorkbook.Sheets(1).Cells(2, col + 1) = Tableau(2)(col)
Next col
For col = LBound(c) To UBound(c)
    ThisWorkbook.Sheets(1).Cells(3, col + 1) = Tableau(3)(col)
Next col
End Sub
```

Dans ce programme, nous avons affecté une matrice à chaque cellule du tableau. Pour le lire, nous devons utiliser, non pas `Tableau(i,j)` car `Tableau` n'a qu'une dimension, mais `Tableau(i)(j)` où `i` désigne une ligne de `Tableau` et `j` la position d'un élément de la matrice insérée dans la ligne `i`. Les matrices commençant par l'indice 0, `Tableau(2)(1)` est égal à 23.

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 qui lient des conditions.

La condition Condition1 AND Condition2 est vraie si Condition1 et Condition2 sont vraies toutes les deux.

La condition Condition1 OR Condition2 est vraie si au moins l'une des deux conditions est vraie.

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 avons 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				

Auteur : Francis Malherbe