

# Utiliser les variables tableaux en VBA Excel

par SilkyRoad ([silkyroad.developpez.com](http://silkyroad.developpez.com))

Date de publication : 14/08/2007

Dernière mise à jour :

Cet article propose une initiation aux variables tableaux, en VBA Excel.

- I - Introduction
- II - Description
  - II-A - Les tableaux de taille fixe
  - II-B - Les tableaux dynamiques
  - II-C - Les tableaux multidimensionnels
- III - L'instruction ReDim
  - III-A - ReDim
  - III-B - Le mot clé Preserve
- IV - Option Base
- V - Les fonctions LBound et UBound
  - V-A - LBound
  - V-B - UBound
- VI - La fonction Array
- VII - La fonction IsArray
- VIII - L'instruction Erase
- IX - Les tableaux de type définis par l'utilisateur
- X - Les tableaux de paramètres ParamArray
- XI - La fonction Filter
- XII - La fonction Join
- XIII - Passer un tableau à une sous-procédure: utilisation de ByRef
- XIV - Quelques exemples
  - XIV-A - Vérifier si un tableau est vide
  - XIV-B - Compter le nombre de doublons dans une plage
  - XIV-C - Alimenter une ListBox multicolonne à partir d'un tableau
  - XIV-D - Trier les données d'un tableau
  - XIV-E - Compter le nombre de dimensions d'un tableau
  - XIV-F - Transférer le contenu d'un tableau dans une feuille de calcul
  - XIV-G - Créer un graphique à partir de variables tableaux
- XV - Liens
- XVI - Téléchargement

## I - Introduction

Les variables tableaux servent à stocker et manipuler des groupes de données du même type.

Les éléments du tableau sont indexés séquentiellement. Chaque élément est identifiable par un numéro d'indice. Les modifications apportées à une donnée du tableau n'affectent pas les autres éléments.

Créer un tableau de taille X revient en quelque sorte à déclarer X variables différentes, en une fois.

Un tableau est constitué d'une ou plusieurs dimensions. Le nombre d'éléments pour chaque dimension est défini par des limites inférieures et supérieures. La taille et les dimensions peuvent être fixes (statiques) ou libres (dynamiques).

Seuls les tableaux dynamiques peuvent modifier leur taille et leur dimension en cours de procédure.

Passer par un tableau n'est pas une obligation. Toutefois, cette méthode permet des gains de temps significatifs, notamment pour la manipulation des grands groupes de données. Sur les grandes collections, il convient d'éviter l'énumération qui est très lente.

Comparez la rapidité d'exécution du code entre:

Vba

```
Dim ObjCell As Range

For Each ObjCell In Range("A1:J65535").Cells
    ObjCell.Value = ObjCell.Value * 2 + 3
Next
```

et

Vba

```
'Exemple issu de la FAQ Excel
'http://excel.developpez.com/faq/?page=Cellule#TabCellulesVariant
'Auteur: Bidou

Dim Montab As Variant, cpt1 As Long, cpt2 As Long

Montab = Range("A1:J65535").Value

For cpt1 = LBound(Montab, 1) To UBound(Montab, 1)
    For cpt2 = LBound(Montab, 2) To UBound(Montab, 2)
```

Vba

```
Montab(cmpt1, cmpt2) = Montab(cmpt1, cmpt2) * 2 + 3
Next cmpt2
Next cmpt1

Range("A1:J65535").Value = Montab
```

Le deuxième code s'exécute environ 20 fois plus vite.

## II - Description

### II-A - Les tableaux de taille fixe

Un tableau est dit de taille fixe lorsque ses dimensions sont prédéfinies au moment de la déclaration de la variable.

La ligne de code suivante montre comment déclarer un tableau fixe.

`Dim NomTableau(2) As String`

L'indice **2** spécifie la taille du tableau. L'indice inférieur d'un tableau peut commencer à 0 ou à 1 en fonction de la définition de l'instruction **Option Base**.

Pour obtenir plus de détails sur la gestion des indices, consultez les chapitres:

- \* Option Base
- \* LBound et UBound

**As String** définit le type de données.

Les tableaux se déclarent de la même façon que les autres variables. Tous **les types de variables** peuvent être utilisés afin de déclarer un tableau. Déclarez explicitement vos tableaux, avec un type de données adapté, afin d'optimiser l'utilisation de l'espace mémoire.

Dans le premier exemple ci-dessous, 0 est le plus petit index du tableau, et 2 l'index le plus élevé. Cela signifie que le tableau pourra contenir 3 éléments.

Cette procédure simplifiée montre comment alimenter les éléments du tableau et comment boucler sur ces mêmes éléments afin d'en lire le contenu.

Vba

```
Option Explicit
Option Base 0

Sub MonPremierTableau()
    'Définit la taille du tableau et le type de données.
    Dim NomTableau(2) As String
    Dim i As Integer
```

Vba

```
'Alimente les éléments du tableau
NomTableau(0) = "a"
NomTableau(1) = "b"
NomTableau(2) = "c"

'Boucle sur les éléments du tableau pour lire leur contenu
For i = 0 To 2
    MsgBox NomTableau(i)
Next i
End Sub
```

## II-B - Les tableaux dynamiques

Si vous ne spécifiez pas la dimension au moment de la déclaration de la variable, le tableau est appelé dynamique.

Ce type de tableau est utilisé lorsque l'on ne connaît pas à l'avance la taille et/ou les dimensions à attribuer. Celles ci seront précisées en cours de procédure, grâce à l'instruction **ReDim**.

Vous pouvez ainsi changer aussi souvent que vous le souhaitez:

- \* Le nombre de dimensions (Voir le chapitre 'Les tableaux multidimensionnels').
- \* Le nombre d'éléments (Voir les chapitres 'ReDim', 'Option Base').
- \* Les limites supérieures et inférieures de chaque dimension (Voir les chapitres 'Option Base', 'LBound et UBound').

Pour rendre un tableau dynamique, n'indiquez aucune valeur entre les parenthèses (contrairement aux tableaux fixes) lorsque vous déclarez la variable.

### Dim NomTableau() As String

Cet exemple montre comment définir la taille du tableau à partir de la variable "i".

Vous remarquerez qu'**Option Base** n'est pas précisé: 0 est donc le plus petit indice (index) du tableau.

Vba

Vba

```
Option Explicit

Sub MonDeuxiemeTableau()
    'Définit le type de données pour le tableau.
    Dim NomTableau() As String
    Dim i As Integer, j As Integer

    i = 2
    'Définit la taille du tableau
    ReDim NomTableau(i)

    'Alimente les éléments du tableau
    For j = 0 To UBound(NomTableau)
        NomTableau(j) = Chr(65 + j)
    Next j

    'Boucle sur les éléments du tableau
    For j = 0 To UBound(NomTableau)
        MsgBox NomTableau(j)
    Next j
End Sub
```

C'est **ReDim** qui attribue de l'espace mémoire aux tableaux dynamiques.

Consultez le chapitre réservé à cette instruction pour plus de détails.

## II-C - Les tableaux multidimensionnels

Tous les exemples vus jusqu'à présent étaient à dimension unique:

`Dim NomTableau(2) As String` 'déclare un tableau fixe

`ReDim LeTableau(i)` 'Redimensionne un tableau dynamique

De la même manière, il est possible de créer des tableaux multidimensionnels statiques et dynamiques.


Vous pouvez déclarer jusqu'à 60 dimensions dans une variable tableau.

Il suffit d'insérer des virgules pour séparer chaque dimension, quand vous déclarez le tableau.

Cet exemple déclare un tableau de 3 dimensions:

`Dim NomTableau(5, 10, 20) As Integer`

Le nombre total d'éléments disponible est donc le produit des tailles de toutes les dimensions.

 *Lorsque vous souhaitez agrandir un tableau dynamique tout en conservant les données existantes, seule la dernière dimension peut être redimensionnée*

*(Voir le chapitre **ReDim Preserve** pour plus de détails).*

Utilisez des boucles imbriquées pour manipuler les tableaux à plusieurs dimensions.

Vba

```
Option Explicit

Sub ExempleTableau_MultiDimensionnel()
    Dim i As Integer, j As Integer
    'Définit le tableau à 2 dimensions ainsi que leur taille.
    Dim VarTab(1 To 3, 1 To 6) As String

    For i = 1 To UBound(VarTab, 1) 'boucle sur la 1ere dimension
        For j = 1 To UBound(VarTab, 2) 'boucle sur la 2eme dimension
            'Alimente les éléments du tableaux
            VarTab(i, j) = i & j
            'Lit les éléments du tableau
            Debug.Print VarTab(i, j)
        Next j
    Next i
End Sub
```

Le code suivant alimente chaque élément du tableau avec des lettres aléatoires, entre A et Z.

Ensuite la macro trie par ordre croissant une des colonnes (au choix de l'utilisateur), dans la 1ere dimension du tableau.

Vba

```
Option Explicit

Sub TriCroissantMulticolonnes()
    'Déclare un tableau à 2 dimensions.
    Dim Tableau(1 To 4, 1 To 50) As String
    Dim i As Integer, j As Integer, y As Integer
    Dim indexColTri As Byte
    Dim t As Variant
    Dim Resultat As String

    '-----Chargement du tableau -----
    'Remplit chaque élément avec des lettres aléatoires, entre A et Z
```



## Vba

```
For i = 1 To UBound(Tableau, 2)
  For j = 1 To UBound(Tableau, 1)
    Randomize
    Tableau(j, i) = Chr(Int((26 * Rnd) + 1) + 64)
  Next j
Next i
'-----

'---- Applique un tri sur une des colonnes du tableau ----
'Choisissez la colonne à trier: 1= 1ere colonne , 2= 2eme colonne...etc ...
indexColTri = 1

'On sort si l'index de colonne indiqué (indexColTri) est plus grand que la taille de la première
'dimension dans le tableau.
If indexColTri > UBound(Tableau, 1) Then Exit Sub

For i = 1 To UBound(Tableau, 2)
  For j = 1 To UBound(Tableau, 2) - 1

    '-----
    'syntaxe pour le tri de données type Date
    'If CDate(Tableau(indexColTri, j)) > CDate(Tableau(indexColTri, j + 1)) Then
    'Pensez à adapter le type de variable: Dim Tableau(1 To 4, 1 To 50) As Date

    'syntaxe pour le tri de données type numérique
    'If CDec(Tableau(indexColTri, j)) > CDec(Tableau(indexColTri, j + 1)) Then
    'Pensez à adapter le type de variable: Dim Tableau(1 To 4, 1 To 50) As Long ...

    'syntaxe pour le tri de données type Texte
    If Tableau(indexColTri, j) > Tableau(indexColTri, j + 1) Then
      '-----

      For y = 1 To UBound(Tableau, 1)
        t = Tableau(y, j)
        Tableau(y, j) = Tableau(y, j + 1)
        Tableau(y, j + 1) = t
      Next y

    End If
  Next j
Next i
'-----

'---- Affiche le résultat dans la fenêtre d'exécution ----
For i = 1 To UBound(Tableau, 2)
  Resultat = ""

  For j = 1 To UBound(Tableau, 1)
    Resultat = Resultat & Tableau(j, i) & vbTab
  Next j

  Debug.Print Resultat
Next i
'-----

End Sub
```

## III - L'instruction ReDim

### III-A - ReDim

L'instruction **ReDim** est utilisée pour définir (ou redéfinir), en cours de procédure, l'espace mémoire alloué à un tableau dynamique.

**ReDim** sert pour:

- \* Redéfinir le nombre d'éléments.
- \* Changer le nombre de dimensions.
- \* Etablir les limites supérieures et inférieures de chaque dimension.

Exemple:

Vba

```
Option Explicit
Option Base 0

Sub Test()
    'Déclare la variable tableau
    Dim NomTableau() As Single
    Dim i As Integer

    'Redéfinit la taille du tableau
    ReDim NomTableau(2)
    'Boucle sur les éléments du tableau pour le remplir
    For i = 0 To UBound(NomTableau)
        NomTableau(i) = (3 + i) / 2
    Next i
End Sub
```

Vous pouvez appliquer l'instruction **ReDim** plusieurs fois dans une même procédure.



*A chaque fois que vous modifiez la taille d'un tableau, le contenu des anciens éléments est effacé.*

Vba

```
Option Explicit
```

Vba

```
Option Base 0

Sub Test()
'Déclare la variable
Dim NomTableau() As String
Dim i As Integer

'Définit la taille du tableau
ReDim NomTableau(5)
'Boucle sur les éléments du tableau pour le remplir
'avec les lettres A,B,C,D,E et F
For i = 0 To UBound(NomTableau)
    NomTableau(i) = Chr(65 + i)
Next i

'Renvoie la lettre A
MsgBox "Premier élément du tableau: " & NomTableau(0)

'Redéfinit et réduit la taille du tableau.
ReDim NomTableau(3)

'Renvoie une chaîne vide: les anciens éléments ont été effacés
'lorsque la taille du tableau a été redéfinie.
MsgBox "Premier élément du tableau: " & NomTableau(0)
End Sub
```

Attention à la saisie du nom de la variable tableau lorsque vous utilisez **ReDim**. L'instruction a une action déclarative si la variable spécifiée n'existe pas formellement. Même si **Option Explicit** est ajouté en tête de module, une erreur de saisie dans le nom ne renverra pas de message d'erreur et un nouveau tableau sera créé.

Vba

```
Option Explicit

Sub Test()
    Dim NomTableau() As Long

    ReDim NomTableau(5)
    'Le "e" a été oublié volontairement pour montrer que contrairement
    'aux variables classiques (scalaires), la procédure ne renvoie pas d'erreur, même
    'si option Explicit à été déclaré: Un deuxième tableau est créé.

End Sub
```

Lorsque vous appliquez **ReDim** sur des tableaux formalisés, vous pouvez modifier le nombre d'éléments mais pas directement le type de données. Si vous souhaitez aussi changer les types de données en cours de procédure, utilisez une variable Variant et la syntaxe suivante:

Vba

```
Sub Test()
```

Vba

```
Dim NomVariable As Variant
Dim i As Integer

'Modifie la variable en tableau de type String.
ReDim NomVariable(1 To 3) As String
'Alimente les éléments en données String
For i = 1 To UBound(NomVariable)
    NomVariable(i) = Chr(64 + i)
Next i

'Modifie la variable en tableau de type Integer.
ReDim NomVariable(1 To 2) As Integer
'Alimente les éléments en données numériques
For i = 1 To UBound(NomVariable)
    NomVariable(i) = i
Next i

End Sub
```

### III-B - Le mot clé Preserve

Nous avons vu que l'instruction **ReDim** modifie la taille des tableaux, mais efface les anciens éléments.

Ajoutez le mot clé **Preserve** pour agrandir un tableau dynamique tout en conservant les valeurs existantes. Vous pourrez ainsi modifier la taille de la dernière dimension d'un tableau sans perdre les données déjà stockées dans les éléments d'origine.

Cet exemple montre comment lister dans un tableau à deux dimensions, le nom des fichiers d'un répertoire et leur date de création ou de dernière modification. Comme le nombre de fichiers n'est pas connu à l'avance, la taille du tableau augmente d'une unité à chaque tour de boucle, sans effacer les enregistrements qu'il contient déjà.

Vous remarquerez que c'est la dernière dimension du tableau (variable x) qui est modifiée.

Vba

```
Option Explicit

Sub ListeFichiersRepertoire()
    Dim Repertoire As String, Fichier As String
    Dim Tableau() As Variant
    Dim x As Integer, i As Integer
    Dim VerifTab As Variant

    'Définit le répertoire pour la recherche
    Repertoire = "C:\Documents and Settings\dossier"
    'Recherche tous les types de fichiers
    Fichier = Dir(Repertoire & "\*.*)"

    'Boucle sur les fichiers pour récupérer les infos
    Do While Fichier <> ""
        'Incrémente le compteur de fichiers
        x = x + 1
    Loop
```

Vba

```
'--- Redéfinit la taille de la dernière dimension du tableau
ReDim Preserve Tableau(1 To 2, 1 To x)
'-----

'Récupère le nom du fichier
Tableau(1, x) = Fichier
'Récupère la date et l'heure de création ou de dernière modification.
Tableau(2, x) = FileDateTime(Repertoire & "\" & Fichier)
Fichier = Dir
Loop

'--- On vérifie si le tableau est vide
On Error Resume Next
'VerifTab va prendre la valeur Empty si le tableau est vide.
VerifTab = UBound(Tableau)
On Error GoTo 0

If IsEmpty(VerifTab) Then Exit Sub
'---

'Boucle pour lire le contenu du tableau.
'UBound(Tableau, 2) permet de récupérer la limite supérieure de la 2eme dimension
For i = 1 To UBound(Tableau, 2)
    'Inscrit le résultat dans la fenêtre d'exécution (Ctrl+G)
    Debug.Print Tableau(1, i) & " --> " & Tableau(2, i)
Next i
End Sub
```

Le mot clé Preserve:

- \* Permet uniquement de modifier la limite supérieure de la dernière dimension du tableau.
- \* Ne permet pas de modifier le nombre de dimensions.

## IV - Option Base

La limite inférieure des tableaux est définie par l'instruction **Option Base**. La valeur peut être 0 ou 1.

La base par défaut est 0 si l'instruction n'est pas spécifiée dans le module.

**Option Base** doit être placée tout en haut du module, avant toute procédure ou déclaration.

Celle ci est valable uniquement pour le module où elle est située.

Vous pouvez vérifier l'action d'**Option Base** en testant les deux codes suivants.

Le premier (base 0) renvoie des limites 0 et 5. Le tableau peut donc contenir 6 éléments.

Vba

```
Option Explicit
Option Base 0

Sub Test_Base()
    Dim NomTableau(5) As String

    MsgBox "Index inférieur: " & LBound(NomTableau) & vbCrLf & _
        "Index supérieur: " & UBound(NomTableau)
End Sub
```

Le deuxième (base 1) renvoie des limites 1 et 5. Le tableau peut donc contenir 5 éléments.

Vba

```
Option Explicit
Option Base 1

Sub Test_Base()
    Dim NomTableau(5) As String

    MsgBox "Index inférieur: " & LBound(NomTableau) & vbCrLf & _
        "Index supérieur: " & UBound(NomTableau)
End Sub
```

Pour vous affranchir des particularités d'**Option Base**, vous pouvez aussi utiliser la clause **To**, afin de contrôler la plage des indices d'un tableau.

La syntaxe est: NomTableau(LimiteInférieure To LimiteSupérieure).

`Dim NomTableau(1 To 5) As String`

Pour définir un tableau multi dimensionnel:

`Dim NomTableau(1 To 5, 1 To 20) As String`

Cet exemple renvoie des limites 1 et 5:

```
Vba
Option Explicit

Sub Test_Base()
    Dim NomTableau(1 To 5) As String

    MsgBox "Index inférieur: " & LBound(NomTableau) & vbCrLf & _
        "Index supérieur: " & UBound(NomTableau)
End Sub
```

## V - Les fonctions LBound et UBound

Les fonctions **LBound** et **UBound** permettent de déterminer la taille d'une dimension dans un tableau.

Leur principe d'utilisation est identique.

### V-A - LBound

**LBound** Renvoie le plus petit indice disponible pour la dimension indiquée.

[MsgBox LBound\(NomTableau, 2\)](#)

Ici, la procédure affiche la limite inférieure de la 2eme dimension.

Utilisez [LBound\(NomTableau, 3\)](#) pour la 3eme dimension ...etc...

1 sera la valeur par défaut si l'argument dimension n'est pas spécifié. Pour tester la première dimension (ou un tableau à dimension unique) vous pouvez donc écrire:

Vba

```
Option Explicit
Option Base 0

Sub Test_LBound()
    Dim NomTableau() As Single

    ReDim NomTableau(8)

    'Affiche la taille inférieure d'un tableau à taille unique.
    'LBound renvoie 0 car "Option Base 0" est indiqué en tête de module.
    MsgBox LBound(NomTableau, 1)
    'ou plus simplement:
    MsgBox LBound(NomTableau)
End Sub
```

La limite inférieure d'une dimension peut être:

\* 0 ou 1, en fonction de la valeur de l'instruction **Option Base** (Consultez le chapitre Option Base pour plus de détails).

\* N'importe quelle valeur pour les dimensions définies à l'aide de la clause [To](#).



**LBound** provoque une erreur si les dimensions des tableaux n'ont pas été initialisées.

```
Vba

Option Explicit
Option Base 0

Sub Test_LBound()
    Dim Tab_x() As Long
    Dim Tab_y(1 To 20, 5 To 30) As Integer
    Dim Tab_z(10) As String

    ReDim Tab_x(5)
    Debug.Print LBound(Tab_x) 'Renvoie 0

    Debug.Print LBound(Tab_y) 'Renvoie 1
    Debug.Print LBound(Tab_y, 2) 'Renvoie 5

    Debug.Print LBound(Tab_z) 'Renvoie 0

    ReDim Tab_x(4 To 8, 1 To 10, 1 To 20)
    Debug.Print LBound(Tab_x) 'Renvoie 4
    Debug.Print LBound(Tab_x, 3) 'Renvoie 1
End Sub
```

## V-B - UBound

**UBound** Renvoie l'indice le plus élevé disponible pour la dimension indiquée.

`MsgBox UBound(NomTableau, 2)`

Ici, la procédure affiche la limite supérieure de la 2eme dimension.

Utilisez `UBound(NomTableau, 3)` pour la 3eme dimension ...etc...

1 sera la valeur par défaut si l'argument dimension n'est pas spécifié. Pour tester la première dimension (ou un tableau à dimension unique) vous pouvez donc écrire:

```
Vba

Option Explicit
Option Base 0

Sub Test_UBound()
```

Vba

```
Dim NomTableau() As Single

ReDim NomTableau(8)

'Affiche la taille inférieure d'un tableau à dimension unique.
'UBound renvoie 8
MsgBox UBound(NomTableau, 1)
'ou plus simplement:
MsgBox UBound(NomTableau)
End Sub
```

**UBound** provoque une erreur si les dimensions des tableaux n'ont pas été initialisées.

D'autres tests pour la fonction **UBound**:

Vba

```
Option Explicit
Option Base 0

Sub Test_UBound()
    Dim Tab_x() As Long
    Dim Tab_y(1 To 20, 5 To 30) As Integer
    Dim Tab_z(10) As String

    ReDim Tab_x(5)
    Debug.Print UBound(Tab_x) 'Renvoie 5

    Debug.Print UBound(Tab_y) 'Renvoie 20
    Debug.Print UBound(Tab_y, 2) 'Renvoie 30

    Debug.Print UBound(Tab_z) 'Renvoie 10

    ReDim Tab_x(4 To 8, 1 To 10, 1 To 20)
    Debug.Print UBound(Tab_x) 'Renvoie 8
    Debug.Print UBound(Tab_x, 3) 'Renvoie 20
End Sub
```

## VI - La fonction Array

La fonction **Array** permet de créer une liste d'éléments, séparés par des virgules. L'utilisation est très diversifiée comme le montre ces quelques exemples:

### Vba

```
'Envoi du classeur actif par mail, à plusieurs destinataires.  
ActiveWorkbook.SendMail Recipients:=Array("mimi@provider.com", "loulou@provider.fr"), _  
    Subject:="Rapport de visite " & ActiveWorkbook.Name, ReturnReceipt:=True
```

### Vba

```
'Copier quelques feuilles du classeur:  
'Crée une copie des feuilles cibles dans un nouveau classeur  
Worksheets(Array("Feuil2", "Feuil3")).Copy
```

### Vba

```
'Boucler sur quelques feuilles du classeur  
  
Dim Ws As Worksheet  
  
For Each Ws In ThisWorkbook.Worksheets(Array("Feuil5", "Feuil7", "Feuil8"))  
    'Inscrit la valeur 1 dans chacune des feuilles cibles  
    Ws.Range("A1").Value = 1  
Next Ws
```

### Vba

```
'Boucler sur quelques contrôles dans un UserForm  
  
Private Sub CommandButton1_Click()  
    Dim Ctrl As Variant  
    Dim j As Byte  
  
    For Each Ctrl In Array(TextBox1, TextBox3, TextBox5)  
        j = j + 1  
        Ctrl.Object.Value = "Champ" & j  
    Next  
End Sub
```

Lorsque vous attribuez le tableau **Array** à une variable, celle ci doit impérativement être de type Variant.

### Vba

```
Option Explicit  
Option Base 0  
  
Sub ExempleArray_V01()  
    Dim NomTableau As Variant  
  
    NomTableau = Array("a", "b", "c")  
End Sub
```

Vba

```
MsgBox NomTableau(0) 'Renvoie "a"  
MsgBox NomTableau(2) 'Renvoie "c"  
End Sub
```

L'exemple suivant montre comment boucler sur le tableau:

Vba

```
Sub Exemple_BoucleArray()  
Dim NomTableau As Variant  
Dim i As Integer  
  
NomTableau = Array("a", "b", "c")  
  
For i = LBound(NomTableau) To UBound(NomTableau)  
    MsgBox NomTableau(i)  
Next i  
End Sub
```

Pour que l'instruction **Option Base** n'ait pas d'influence sur la fonction **Array**, utilisez la syntaxe **VBA.Array**:

Vba

```
Option Explicit  
Option Base 1  
  
Sub ExempleArray_V01()  
Dim NomTableau As Variant  
  
NomTableau = VBA.Array("a", "b", "c")  
  
MsgBox NomTableau(0) 'Renvoie "a"  
MsgBox NomTableau(2) 'Renvoie "c"  
End Sub
```

## VII - La fonction IsArray

La fonction **IsArray** vérifie si une variable est un tableau. La valeur **True** est renvoyée quand c'est le cas.

```
Vba
Sub VerifieExistenceTableau()
    Dim Tableau(2) As String
    Dim NomTableau As Variant

    MsgBox IsArray(Tableau()) 'Renvoie True
    MsgBox IsArray(NomTableau) 'Renvoie False

    NomTableau = Range("A1")
    MsgBox IsArray(NomTableau) 'Renvoie False

    NomTableau = Range("A1:B5")
    MsgBox IsArray(NomTableau) 'Renvoie True
End Sub
```

Cette fonction est très pratique pour contrôler les variables Variant (**NomTableau** dans l'exemple précédent), lorsque celles-ci peuvent prendre en mémoire une donnée spécifique (Range("A1")) ou un tableau (Range("A1:B5")).

Un autre exemple qui gère la multi-sélection de fichiers depuis la boîte de dialogue "Ouvrir".

```
Vba
Dim Fichiers As Variant
Dim i As Integer

'Affiche la boîte dialogue "Ouvrir" en filtrant sur les classeurs Excel
'(C'est l'argument True qui autorise la multi-sélection)
Fichiers = Application.GetOpenFilename("Fichiers Excel (*.xls), *.xls", , , , True)

'Boucle sur le tableau pour récupérer le nom du ou des classeurs sélectionnées.
'(IsArray(Fichiers) renvoie False si aucun fichier n'a été sélectionné).
If IsArray(Fichiers) Then
    For i = 1 To UBound(Fichiers)
        MsgBox Fichiers(i)
    Next
End If
```

## VIII - L'instruction Erase

L'instruction **Erase** réinitialise les tableaux de taille fixe et libère l'espace mémoire réservé aux tableaux dynamiques.

Vba

```

Sub Test_ReinitialisationTableauFixe()
    Dim NomTableau(1 To 3) As String

    'Alimente le tableau
    NomTableau(1) = "a"
    NomTableau(2) = "b"
    NomTableau(3) = "c"

    'Affiche le contenu du 3eme item ("c")
    MsgBox NomTableau(3)

    'Efface le contenu du tableau
    Erase NomTableau

    'Affiche le contenu du 3eme item (renvoie une chaîne vide)
    MsgBox NomTableau(3)
End Sub
    
```

**Erase** ne libère pas d'espace mémoire dans les tableaux de taille fixe.

Voici un récapitulatif des valeurs prises par les variables tableau, après avoir utilisé l'instruction:

Type de variable dans le Tableau fixe	Valeur prise par chaque élément après Erase
Numérique	0
Date	0
Chaîne (longueur variable)	Chaîne vide ""
Chaîne (longueur fixe)	0
Variant	Empty
Objet	Nothing
Type défini par l'utilisateur	Fonction de chaque variable constitutive

**Erase** libère la mémoire utilisée par les tableaux dynamiques.

Ensuite, en cas de réutilisation, vous devez réappliquer l'instruction **ReDim** afin de spécifier les nouvelles dimensions du tableau.

Vba

Vba

```
Sub Test_EraseTableauDynamique()  
Dim NomTableau() As Long  
Dim x As Integer, i As Integer  
  
x = 3  
'Redimensionne le tableau  
ReDim NomTableau(1 To x)  
  
'Alimente le tableau (12,14,16)  
For i = 1 To x  
    NomTableau(i) = (i + 5) * 2  
Next i  
  
'Affiche la valeur d'index la plus élevée du tableau et son contenu  
'(UBound(NomTableau) doit renvoyer la valeur x)  
MsgBox "élément " & UBound(NomTableau) & ": " & NomTableau(UBound(NomTableau))  
  
'libère la mémoire utilisée par le tableau  
Erase NomTableau  
  
x = 2  
'Redimensionne le tableau avant une nouvelle utilisation  
ReDim NomTableau(1 To x)  
  
'Alimente le tableau (18,21)  
For i = 1 To x  
    NomTableau(i) = (i + 5) * 3  
Next i  
  
'Affiche la valeur d'index la plus élevée du tableau et son contenu  
'(UBound(NomTableau) doit renvoyer la valeur x)  
MsgBox "élément " & UBound(NomTableau) & ": " & NomTableau(UBound(NomTableau))  
  
End Sub
```

Il est possible d'appliquer l'instruction **Erase** sur plusieurs tableaux, en une seule fois:

Vba

```
Erase NomTableau_01, NomTableau_02
```

## IX - Les tableaux de type définis par l'utilisateur

Un tableau de type défini par l'utilisateur contient un ou plusieurs éléments de tout type de données, identifiables par des noms "conviviaux". Bien que cette instruction soit un peu plus complexe à mettre en oeuvre, vous y gagnerez en clarté pour les projets volumineux et lors de la manipulation des tableaux aux dimensions multiples.

Vba

```
Option Explicit

'Définition du type
Type InformationsClient
    Matricules(1 To 3) As Long
    NomClient As String
    DateCreation As Date
End Type

Sub Test()
    'Définit le tableau d'enregistrement
    Dim NouvellesDonnees() As InformationsClient
    Dim x As Integer

    'Définit la taille du tableau
    x = 3
    ReDim NouvellesDonnees(1 To x)

    'Alimente le premier élément du tableau: NouvellesDonnees(1)
    NouvellesDonnees(1).Matricules(1) = 2233601
    NouvellesDonnees(1).Matricules(2) = 2233602
    NouvellesDonnees(1).Matricules(3) = 2233608
    NouvellesDonnees(1).DateCreation = CDate("23/07/2005")
    NouvellesDonnees(1).NomClient = "Rififi"

    'Lit quelques infos dans le 1er enregistrement du tableau
    MsgBox "Société " & NouvellesDonnees(1).NomClient & vbCrLf & _
        "Deuxième matricule: " & NouvellesDonnees(1).Matricules(2)
End Sub
```

Vous trouverez un exemple complet dans cet autre tutorial: **Un complément FileSearch pour Excel 2007**.



## X - Les tableaux de paramètres ParamArray

Les tableaux de paramètres permettent de passer un tableau d'arguments à une procédure. Le nombre d'éléments indiqué peut être variable.

### ParamArray:

- \* Doit obligatoirement être déclaré en type Variant.
- \* Ne peut pas coexister avec des arguments Optional.
- \* Doit être placé en dernière position dans la liste des arguments.
- \* Les arguments ne sont pas facultatifs.
- \* Chaque argument peut être d'un type de données différent.
- \* Le nombre d'arguments n'est pas limité.
- \* L'indice inférieur du tableau est toujours 0.

Lorsque l'appel de procédure est effectué, chaque argument fourni dans l'appel devient un élément indexé du tableau.

Par exemple:

```
Vba

Option Explicit

Sub Test()
    'Les 4 derniers arguments ("_Entete", 10, 11, 12) vont être passés
    'dans le tableau de paramètres "VarTableau".
    NomProcédure "Serie", "_Entete", 10, 11, 12
End Sub

Sub NomProcédure(Prefixe As String, ParamArray VarTableau())
    Dim i As Integer

    'Boucle sur les éléments du tableau
    For i = 0 To UBound(VarTableau())
        'Ecrit dans la fenêtre d'exécution (CTRL+G)
        Debug.Print Prefixe & VarTableau(i)
    Next i
End Sub
```

Un autre exemple qui supprime une série de caractères dans une chaîne:

```

Vba

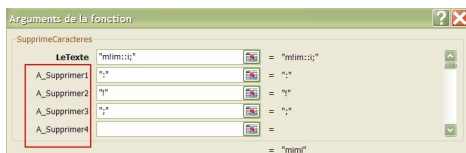
Option Explicit

Sub Test()
    Dim strChaine As String

    strChaine = "m!im:i;"
    'La procédure va supprimer les caractères ; : !
    'dans la variable "Chaine"
    strChaine = SupprimeCaracteres(strChaine, ";", ":", "!")
    MsgBox strChaine
End Sub

Function SupprimeCaracteres(LeTexte As String, ParamArray A_Supprimer1())
    Dim i As Integer
    'Boucle sur les éléments du tableau
    For i = 0 To UBound(A_Supprimer1())
        'Supprime les caractères spécifiés
        LeTexte = Replace(LeTexte, A_Supprimer1(i), "")
    Next i
    SupprimeCaracteres = LeTexte
End Function
    
```

Si vous utilisez **ParamArray** dans une fonction, ajoutez le chiffre 1 en fin de nom du paramètre. Ainsi l'assistant de fonctions Excel incrémentera automatiquement chaque nouvel élément indiqué par l'utilisateur.



## XI - La fonction Filter

Cette fonction permet de filtrer le contenu d'un tableau à une dimension.

**Filter** renvoie un tableau répondant aux critères de filtre spécifiés.

**Filter(sourcesrray, match[, include[, compare]])**

**Sourcesrray** est le tableau de chaînes à une dimension dans lequel la recherche doit être effectuée.

**Match** est la chaîne à rechercher.

**Include:**

Attribuez la valeur True pour que la fonction renvoie les éléments qui contiennent la chaîne **match** spécifiée.

Attribuez la valeur False pour que la fonction renvoie les éléments qui ne contiennent pas la chaîne **match** spécifiée.

**Compare** indique le type de comparaison de chaîne à utiliser:

Constante	Valeur	Description
vbBinaryCompare	0	Les comparaisons distinguent les majuscules des minuscules.
vbTextCompare	1	Les comparaisons ne distinguent pas les majuscules des minuscules: <b>Valeur par défaut si l'argument n'est pas spécifié.</b>
vbDatabaseCompare	2	Microsoft Access seulement. Effectue une comparaison basée sur des informations contenues dans votre base de données.

### Vba

```

Dim DataString(3) As String
Dim InString() As String
Dim i As Integer

'Remplissage du tableau qui va être testé
DataString(0) = "chaîne 1"
DataString(1) = "string 2"
DataString(2) = "chaîne 3"
DataString(3) = "string 4"

'"DataString" est le tableau dans lequel doit être effectué la recherche.
'"str" est la chaîne à rechercher.
    
```

## Vba

```
'La fonction Filter renvoie le tableau "InString" contenant les éléments  
'qui répondent aux critères de la recherche.  
InString = Filter(DataString, "str", True)  
  
'Boucle sur le tableau afin de visualiser les éléments  
'qui répondent aux critères de recherche.  
For i = 0 To UBound(InString)  
    Debug.Print InString(i)  
Next i
```

## XII - La fonction Join

Utilisez la fonction **Join** pour créer une jonction entre les chaînes contenues dans un tableau.

### Join(sourcearray[, delimiter])

**Sourcearray** est le tableau à une dimension contenant les chaînes à joindre.

**Delimiter** (Facultatif) spécifie le séparateur à utiliser entre les éléments du tableau, lors de la jonction. Les éléments sont séparés par un espace si l'argument **Delimiter** n'est pas spécifié.

#### VB6-VBA

```
Dim Tableau(2) As String

Tableau(0) = "Chaine01"
Tableau(1) = "Chaine02"
Tableau(2) = "Chaine03"

'Regroupe les chaînes du tableau, séparés par un point virgule ";"
MsgBox Join(Tableau, ";")

'Regroupe les chaînes du tableau sans séparateur
MsgBox Join(Tableau, "")

'Les éléments sont séparés par un espace si l'argument Delimiter n'est
'pas spécifié.
MsgBox Join(Tableau)

'Regroupe les chaînes du tableau et ajout d'un saut de ligne
'entre chaque élément.
MsgBox Join(Tableau, vbCrLf)
```

## XIII - Passer un tableau à une sous-procédure: utilisation de ByRef

Utilisé comme argument dans une sous-procédure, un tableau doit impérativement être passé par référence (**ByRef**).

L'option par valeur (**ByVal**) provoque une erreur si vous essayer de passer un tableau complet. Les éléments d'un tableau peuvent être passés individuellement avec l'option **ByVal**, mais l'ensemble du tableau doit être géré avec l'option par référence.

Vba

```
Option Explicit

Sub Test()
    Dim NomTableau(1 To 4) As String
    Dim i As Integer

    'Remplit le tableau des caractères A,B,C et D
    For i = 1 To UBound(NomTableau)
        NomTableau(i) = Chr(64 + i)
    Next i

    MsgBox MaFonction(2, NomTableau)
End Sub

'Vous obtiendrez un message d'erreur si vous remplacez ByRef par ByVal.
'ByRef est la valeur par défaut, vous n'êtes donc pas obligé de l'indiquer explicitement.
Function MaFonction(x As Double, ByRef VarTab() As String) As String
    Dim j As Integer
    Dim Resultat As String

    For j = 1 To UBound(VarTab)
        'Répète les caractères du tableau x fois.
        Resultat = Resultat & Application.WorksheetFunction.Rept(VarTab(j), x)
    Next j

    MaFonction = Resultat
End Function
```

Remarques:

Dans cet exemple, VarTab et NomTableau sont déclarés avec le même type de données (String) sinon une erreur 'Incompatibilité de type' survient.

Lorsque vous transmettez un tableau par référence (**ByRef**) à une sous-procédure, celui ci ne peut être redimensionné au sein de la sous-procédure.

Si vous devez utiliser un tableau **ByVal** afin d'empêcher que les modifications des éléments dans la sous-procédure ne soient répercutées sur le tableau appelant:

Transférez le tableau dans une variable type Variant (Plus généralement, cette technique permet d'affecter rapidement la totalité d'un tableau à un autre tableau).

Ensuite, passez cette variable Variant à l'argument **ByVal**.

L'exemple suivant renvoie le x ième rang d'un tableau aléatoire trié par ordre croissant, sans affecter le tableau appelant.

Vba

```
Option Explicit

Sub Test()
    Dim NomTableau(1 To 10) As Integer
    Dim VarTemp As Variant
    Dim i As Integer

    'Initialisation du générateur de nombres aléatoires
    Randomize

    'Remplit le tableau de valeurs aléatoires, entre 1 et 50
    For i = 1 To UBound(NomTableau)
        NomTableau(i) = Int((50 * Rnd) + 1)
    Next i

    'Transfère le tableau dans la variable type Variant
    VarTemp = NomTableau()

    MsgBox Rang_TableauTrie(1, VarTemp)
End Sub

'Renvoie le x ième rang d'un tableau aléatoire trié par ordre croissant
Function Rang_TableauTrie(x As Double, ByVal VarTab As Variant) As Integer
    Dim i As Integer, m As Integer, k As Integer
    Dim ValTemp As Integer

    'Vérifie si le rang demandé n'est pas plus grand que le nombre d'éléments dans le tableau
    If x > UBound(VarTab) Then Exit Function

    'Applique un tri croissant sur le tableau
    For i = 1 To UBound(VarTab)
        m = i

        For k = m + 1 To UBound(VarTab)
            If VarTab(k) <= VarTab(m) Then m = k
        Next k

        If i <> m Then
            ValTemp = VarTab(m): VarTab(m) = VarTab(i): VarTab(i) = ValTemp
        End If
    Next i

    Rang_TableauTrie = VarTab(x)
End Function
```

Vba

End Function



## XIV - Quelques exemples

### XIV-A - Vérifier si un tableau est vide

Il est parfois utile de contrôler si un tableau a été initialisé. Le code proposé ici permet cette vérification.

La fonction **IsEmpty** renvoie la valeur True si l'expression (impérativement de type Variant) qui lui est appliquée n'est pas initialisée ou contient une valeur de type **Empty** (vide).

Vba

```
Dim Tableau() As Long
'La variable VarTab doit impérativement être de type Variant.
Dim VarTab As Variant

'
'...La procédure
'

On Error Resume Next
'VarTab va prendre la valeur Empty si le tableau est vide.
VarTab = UBound(Tableau)
On Error GoTo 0

If IsEmpty(VarTab) Then MsgBox "Le tableau est vide"
```

### XIV-B - Compter le nombre de doublons dans une plage

Ce code liste et compte les informations qui se répètent dans une plage de cellules.

Dans cet exemple la plage de cellules est limitée à une colonne.

Vba

```
Option Explicit
Option Base 1

Sub listeDoublonsPlage()
    Dim Plage As Range
    Dim Tableau(), Resultat() As String
    Dim i As Integer, j As Integer, m As Integer
    Dim Un As Collection
    Dim Doublons As String

    Set Un = New Collection
    'La plage de cellules (sur une colonne) à tester
    Set Plage = Range("A1:A" & Range("A65536").End(xlUp).Row)

    Tableau = Plage.Value
```

Vba

```
On Error Resume Next
'boucle sur la plage à tester
For i = 1 To Plage.Count

    ReDim Preserve Resultat(2, m + 1)

    'Utilise une collection pour rechercher les doublons
    '(les collections n'acceptent que des données uniques)
    Un.Add Tableau(i, 1), CStr(Tableau(i, 1))

    'S'il y a une erreur (donc présence d'un doublon)
    If Err <> 0 Then

        'boucle sur le tableau des doublons pour vérifier s'il a déjà
        'été identifié
        For j = 1 To m + 1
            'Si oui, on incrémente le compteur
            If Resultat(1, j) = Tableau(i, 1) Then
                Resultat(2, j) = Resultat(2, j) + 1
                Err.Clear
                Exit For
            End If
        Next j

        'Si non, on ajoute le doublon dans le tableau
        If Err <> 0 Then
            Resultat(1, m + 1) = Tableau(i, 1)
            Resultat(2, m + 1) = 1

            m = m + 1
            Err.Clear

        End If
    End If
Next i

'----- Affiche la liste et le nombre de doublons -----
For j = 1 To m
    Doublons = Doublons & Resultat(1, j) & " --> " & _
                Resultat(2, j) & vbCrLf
Next j

MsgBox Doublons

Set Un = Nothing
End Sub
```

## XIV-C - Alimenter une ListBox multicolonne à partir d'un tableau

Le code récupère le contenu d'une plage de cellules dans une variable et transfère les données dans une ListBox.

Vba

```
Option Explicit

Private Sub UserForm_Initialize()
    Dim TabTemp As Variant
```

Vba

```
'Chargement d'une plage de cellules dans la variable TabTemp
TabTemp = Range("A1:C10").Value

'Définit le nombre de colonnes pour la ListBox.
ListBox1.ColumnCount = UBound(TabTemp)
'Chargement du tableau dans la ListBox
ListBox1.List() = TabTemp
End Sub
```

Alimenter un tableau et une ListBox depuis un recordset Access:

Cette macro Excel crée une requête dans une table Access et affiche le résultat dans une ListBox.

Nécessite d'activer la référence "Microsoft ActiveX Data Objects x.x Library".

Vba

```
Option Explicit

Private Sub CommandButton1_Click()
Dim Cn As ADODB.Connection
Dim Rs As ADODB.Recordset
Dim i As Integer, N As Integer
Dim Fichier As String
Dim Tbl() As Variant

'Définit le chemin de la base Access
Fichier = "C:\Documents and Settings\mimi\dossier\dataBase.mdb"

'Connection à la base
Set Cn = New ADODB.Connection
Cn.Open "Provider=Microsoft.Jet.OLEDB.4.0; Data Source=" & _
Fichier & ";"
Set Rs = New ADODB.Recordset

'Requête
With Rs
.ActiveConnection = Cn
.Open "SELECT * FROM NomTable" _
, , adOpenStatic, adLockOptimistic, adCmdText
End With

'efface les données existantes dans la ListBox
ListBox1.Clear
'Redéfinit le nombre de colonnes dans la listbox en fonction du
nombre de champs contenu dans le Recordset.
ListBox1.ColumnCount = Rs.Fields.Count

'Redimensionne le tableau en fonction du nombre de champs et d'enregistrements
contenu dans le recordset.
ReDim Tbl(Rs.RecordCount - 1, Rs.Fields.Count)

'Boucle sur les enregistrements pour les passer dans le tableau.
Do While Not Rs.EOF
```

Vba

```
For i = 0 To Rs.Fields.Count - 1
    If IsNull(Rs.Fields(i).Value) = True Then
        Tbl(N, i) = ""
    Else
        Tbl(N, i) = CStr(Rs.Fields(i).Value)
    End If
Next i

N = N + 1
Rs.MoveNext
Loop

'Transfert le tableau dans la ListBox
ListBox1.List() = Tbl

'Ferme le RecordSet
Rs.Close
Set Rs = Nothing
'Ferme la connexion
Cn.Close
Set Cn = Nothing
End Sub
```

## XIV-D - Trier les données d'un tableau

Ce code récupère les données d'une plage de cellules (contenant des données numériques) et tri les valeurs du tableau par ordre décroissant. Pour obtenir la liste par ordre croissant, il suffira de boucler sur le tableau, du dernier élément vers le premier.

Vba

```
Option Explicit

Sub Tri_Tableau()
    Dim Valeur As Integer
    Dim i As Integer
    Dim Cible As Variant
    Dim Tableau() As Single

    'Remplissage tableau avec la plage de cellules A1:A10
    ReDim Tableau(0 To 9)
    For i = 0 To UBound(Tableau())
        Tableau(i) = Cells(i + 1, 1)
    Next i

    Do 'tri décroissant
        Valeur = 0

        For i = 0 To UBound(Tableau) - 1
            If Tableau(i) < Tableau(i + 1) Then
                Cible = Tableau(i)
                Tableau(i) = Tableau(i + 1)
                Tableau(i + 1) = Cible
                Valeur = 1
            End If
        Next i
    Loop While Valeur = 1
```

## Vba

```
'vérification du tri décroissant
For i = 0 To UBound(Tableau)
    Debug.Print Tableau(i)
Next i

'Pour obtenir un ordre croissant, il suffit de boucler du dernier élément
'du tableau, vers le premier.

'For i = UBound(Tableau) To 0 Step -1
    'Debug.Print Tableau(i)
'Next i
End Sub
```

Un autre exemple qui alimente le tableau à partir de données numériques aléatoires et effectue un tri par ordre croissant.

## Vba

```
Option Explicit
Option Base 1

Sub TriCroissantTableau()
    Dim TabSynthese(20) As Integer
    Dim i As Integer, x As Integer, k As Integer
    Dim ValTemp As Integer

    For i = 1 To 20
        'Initialisation du générateur de nombres aléatoires, entre 1 et 100
        Randomize
        'Alimente le tableau avec la valeur aléatoire
        TabSynthese(i) = Int((100 * Rnd) + 1)
    Next i

    'Applique un tri croissant sur le tableau
    For i = 1 To UBound(TabSynthese)
        x = i

        For k = x + 1 To UBound(TabSynthese)
            If TabSynthese(k) <= TabSynthese(x) Then x = k
        Next k

        If i <> x Then
            ValTemp = TabSynthese(x): TabSynthese(x) = TabSynthese(i): TabSynthese(i) = ValTemp
        End If
    Next i

    'contrôle le résultat du tri dans la fenêtre d'execution
    For i = 1 To UBound(TabSynthese)
        Debug.Print TabSynthese(i)
    Next i
End Sub
```

```
Vba  
End Sub
```

Vous trouverez un exemple complet de tri dans cet autre tutoriel: **Un complément FileSearch pour Excel 2007.**

## XIV-E - Compter le nombre de dimensions d'un tableau

Utilisez cette procédure pour vérifier le nombre de dimensions que contient un tableau.

```
Vba  
  
Option Base 0  
  
Sub Test()  
    Dim Tableau() As Integer  
  
    ReDim Tableau(2, 4, 8)  
  
    MsgBox NombreDimensions(Tableau)  
End Sub  
  
Function NombreDimensions(varTab As Variant) As Integer  
    Dim Compteur As Integer  
  
    If Not IsArray(varTab) Then Exit Function  
  
    On Error GoTo Fin  
  
    Do  
        Compteur = Compteur + 1  
        Debug.Print UBound(varTab, Compteur)  
    Loop  
  
Fin:  
    NombreDimensions = Compteur - 1  
End Function
```

## XIV-F - Transférer le contenu d'un tableau dans une feuille de calcul

Cette macro copie une variable tableau dans la feuille de calcul.

```
Vba  
  
Option Explicit  
  
Sub TransfertTableau_FeuilleExcel()
```

Vba

```
Dim x As Integer, y As Integer
Dim i As Integer, j As Integer
Dim NomTableau() As String

'Redéfinit la taille du tableau
x = 10
y = 5
ReDim NomTableau(1 To x, 1 To y)

'Alimente les éléments du tableau
For i = 1 To x
    For j = 1 To y
        NomTableau(i, j) = i & "-" & j
    Next j
Next i

'Transfère les éléments du tableau dans la feuille de calcul
Range(Cells(1, 1), Cells(UBound(NomTableau, 1), UBound(NomTableau, 2))) = NomTableau
End Sub
```

## XIV-G - Créer un graphique à partir de variables tableaux

La première partie de la procédure permet de remplir deux tableaux.

Ces tableaux sont ensuite utilisés pour alimenter les valeurs d'ordonnées et d'abscisses.

Vba

```
Option Explicit

Sub creationGraphiqueParTableaux()
    Dim i As Byte
    Dim TabAbscisses(1 To 10) As Integer, TabOrdonnees(1 To 10) As Integer

    'Initialise le générateur de nombres aléatoires.
    Randomize

    For i = 1 To 10
        'Alimente le tableau pour les abscisses.
        TabAbscisses(i) = i

        'Alimente le tableau pour les ordonnées.
        'Le Tableau est rempli par des valeurs aléatoires (entre 1 et 50).
        TabOrdonnees(i) = Int((50 * Rnd) + 1)
    Next i

    'Création graphique
    Charts.Add
    'Définit la localisation du graphique:
    'dans la feuille de calcul (Feuill pour cet exemple)
    ActiveChart.Location _
    Where:=xlLocationAsObject, Name:="Feuill1"

    'Ajoute une série dans le graphique
    With ActiveChart
```

Vba

```
.SeriesCollection.NewSeries  
.SeriesCollection(1).XValues = TabAbscisses() 'Abscisses  
.SeriesCollection(1).Values = TabOrdonnees() 'Ordonnées  
'Définit le type (Courbe)  
.ChartType = xlLine  
.SeriesCollection(1).Name = "Série aléatoire"  
End With  
End Sub
```

Remarque:

Cette méthode est limitée par le nombre de caractères que vous pourrez insérer dans la barre de formules (La fonction qui s'affiche lorsque vous sélectionnez une série dans le graphique).

Par exemple:

```
=SERIE(0.1.2.3.4.5.6.7.8.9.10);{36.13.10.45.21.44.40.19.49.44};1)
```

Cette limite est d'environ 450 caractères.



## XV - Liens

### Utiliser les variables en VBA Excel

**You may receive a "Run-time error 1004" error message when you programmatically set a large array string to a range in Excel 2003.**

## XVI - Téléchargement

