Vk

Zeste de savoir

Les tableaux en VBA

jeudi 07 novembre 2024

Table des matières

Introduction . . . 1
1. Déclaration et première utilisation . . . 2
1.1. Différents modes de déclaration . . . 2
1.2. Affectation . . . 3
1.3. Passage de tableaux en paramètres . . . 4
2. Type, dimensions, limites et itération . . . 5
2.1. Type de données du tableau . . . 5
2.2. Dimensions . . . 6
2.3. Limites . . . 8
2.4. Redimensionnement . . . 10
2.5. Itération . . . 11
3. Autres façons de créer un tableau . . . 12
3.1. Affectation d'un tableau dynamique à partir… . . . 12
3.2. Déclaration en tant que constante . . . 15
4. Opérations supplémentaires . . . 15
4.1. Construire un texte à partir d'un tableau . . . 15
4.2. Réinitialiser un tableau . . . 16
4.3. Transposer un tableau . . . 17
4.4. Calculer le nombre d'éléments par dimension . . . 18
4.5. Lire et écrire des plages . . . 18
5. Nouveaux types et tableaux . . . 19
Conclusion . . . 20

Introduction

Les tableaux, aussi connus sous le nom d'Array en VBA, sont peut-être déjà une notion que vous maîtrisez. Quoiqu'il en soit, nous ne pouvons pas parler de façons d'organiser les données sans les mentionner.

En effet, les tableaux sont notre premier moyen de stocker plusieurs informations dans une seule variable. Par exemple, imaginons que nous préparions nos vacances et que nous souhaitions stocker nos lieux de destination. En recourant à de simples variables, il faudrait associer chaque lieu à une variable : lieu1 = "Paris", lieu2 = "Bretagne", …, lieuX = "Destination finale". Difficile alors de jongler avec toutes ces variables sans en oublier. Avec un tableau, nous pourrions au contraire tout stocker dans une unique variable : lieu = Array("Paris", "Bretagne", …, "Destination finale").

Pour résumer, les tableaux sont une sorte de conteneur séquentiel de valeurs ayant le même type. Chaque valeur se trouve dans une case et chaque case a une position indexée. De cette manière, il est possible d'accéder à toutes les valeurs avec une seule variable.

Les tableaux vont ainsi simplifier notre maîtrise des données. De plus, nous verrons qu'ils vont aussi nous être utiles dans la lecture et l'écriture des plages de données.

Ce billet va donc présenter ou rappeler le fonctionnement des tableaux en VBA.

1. Déclaration et première utilisation

Dans cette première section, nous allons voir comment déclarer des tableaux en VBA pour commencer à nous en servir.

Les tableaux les plus simples ont une unique dimension :

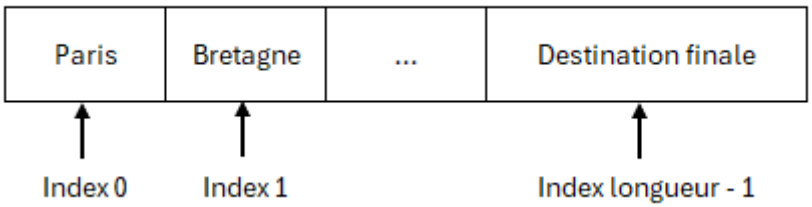

FIGURE 1.1. – Représentation tableau 1D (Option Base 0).

Tout cela n'a pas l'air si différent des cellules dans une feuille de calcul, n'est-ce pas ?

1.1. Différents modes de déclaration

Nous pouvons distinguer les tableaux dits fixes des tableaux dits dynamiques.

Dans tous les cas, ils se déclarent comme à l'accoutumée en utilisant les instructions `Dim`, `Public`, `Private` ou encore `Static`.

1.1.1. Fixe

Comme son nom l'indique, un tableau fixe aura une taille fixe. Il suffit d'indiquer au moins les limites supérieures pour les dimensions voulues entre parenthèses pour le créer. Les valeurs contenues sont alors initialisées par défaut selon le type (0 pour un nombre, "" pour les chaînes ou encore vide pour les variants par exemple).

Voici des exemples simples pour commencer en renseignant la limite supérieure :

```
1 ' Tableau fixe d'entiers à 1 dimension de 3 éléments
2 Dim iTableauFixeEntier(2) As Integer ' index 0 1 2
3
4 Dim sTableauFixeString(3) As String
5 Dim vTableauFixeVariant(4) As Variant
```

Cela implique qu'un tableau fixe ne pourra pas être redimensionné par la suite. Il faut donc être certain que le nombre de valeurs n'évoluera pas pour privilégier ce type de déclaration.

1.1.2. Dynamique

Au contraire, un tableau dynamique pourra être agrandit ou rétrécit.

C'est un avantage quand nous ne savons pas à l'avance combien de valeurs constitueront le tableau ou bien ou encore si ce nombre variera durant l'existence du tableau. En outre, cela permet de prendre la juste place en mémoire sans allouer des tableaux trop grands (nous pourrions imaginer un tableau fixe avec une taille très grande pour être sûrs de tout stocker).

En reprenant l'exemple des lieux de destination, cela serait un bon choix, car notre nombre de destinations pourrait varier au cours du voyage.

Un tableau dynamique se déclare de la même façon qu'un tableau fixe, sauf qu'il faut omettre les paramètres entre parenthèses :

```
1 ' Tableau dynamique d'entiers
2 Dim iTableauDynamiqueEntier() As Integer '
3
4 Dim sTableauDynamiqueString() As String
5 Dim vTableauDynamiqueVariant() As Variant
```

Notons qu'il est possible d'utiliser une variable de type `Variant` pour déclarer un tableau dynamique à la place d'un tableau de variants.

```
1 ' Variable de type Variant qui peut contenir un tableau de variants
2 Dim vTableauDynamiqueVariant2 As Variant
```

Pour pouvoir être utilisé, un tableau dynamique devra d'abord d'être redimensionné afin de lui allouer de la place en mémoire. C'est une opération que nous détaillerons dans la seconde section de ce billet.

1.2. Affectation

Il est possible d'affecter un tableau en mettant des valeurs à des positions, mais aussi via des fonctions retournant des tableaux telles que la fonction `Array`.

1.2.1. Par position

L'accès à une valeur d'un tableau se fait avec le sucre syntaxique `()` en passant la position voulue. Il est ainsi possible de la lire comme de la modifier :

```
1 Dim iTableauFixe(1) As Integer
2 iTableauFixe(0) = 5
3 Debug.Print (iTableauFixe(0)) ' 5
4 Debug.Print (iTableauFixe(1)) ' 0 (valeur par défaut automatique
      tableau fixe)
```

1.2.2. Fonction Array

La fonction `Array`, utilisée en introduction, permet de retourner un tableau dynamique de variants à partir des valeurs fournies.

```
1 Dim vTableauAvecArray1() As Variant
2
3 vTableauAvecArray1 = Array("Paris", "Bretagne", "...",
      "Destination finale")
4
5 Sheets("Array").Range("A1:D1").Value = vTableauAvecArray1
```

	A	B	C	D	E	F
1	Paris	Bretagne	...	Destination finale		
2						
3						
4						

FIGURE 1.2. – Résultat exemple utilisation fonction Array.

Comme c'est un tableau dynamique, il sera donc possible de le redimensionner.

1.3. Passage de tableaux en paramètres

Nous pouvons passer des tableaux fixes et dynamiques en paramètres de fonctions ou de procédures. Seuls les tableaux dynamiques peuvent être passés par valeur plutôt que par référence. Dans ce cas, il faut alors utiliser la notation variable de type `Variant`.

```vba
Private Sub Exemple_Tableau_Fixe_Par_Reference(ByRef vTableau() As
    Integer)
    vTableau(2) = vTableau(0) + vTableau(1)
End Sub

Private Sub Exemple_Tableau_Dynamique_Par_Valeur(ByVal vTableau As
    Variant)
    vTableau(2) = vTableau(0) + vTableau(1)
End Sub

Private Sub Exemple_Tableau_Dynamique_Par_Reference(ByRef
    vTableau() As Variant)
    vTableau(2) = vTableau(0) + vTableau(1)
End Sub

Private Sub Exemple_Passage_Tableaux()
    Dim vTableauFixe1(2) As Integer
    vTableauFixe1(0) = 1
    vTableauFixe1(1) = 2
    Exemple_Tableau_Fixe_Par_Reference vTableauFixe1
    Debug.Print (vTableauFixe1(2)) ' 3

    Dim vTableauDynamique1() As Variant
    vTableauDynamique1 = Array(1, 2, 0)
    Exemple_Tableau_Dynamique_Par_Valeur vTableauDynamique1
    Debug.Print (vTableauDynamique1(2)) ' 0

    Dim vTableauDynamique2() As Variant
    vTableauDynamique2 = Array(1, 2, 0)
    Exemple_Tableau_Dynamique_Par_Reference vTableauDynamique2
    Debug.Print (vTableauDynamique2(2)) ' 3
End Sub
```

Durant cette section, nous avons déclaré et manipulé nos premiers tableaux en VBA.

2. Type, dimensions, limites et itération

Nous allons maintenant en apprendre davantage sur la construction des tableaux et leur utilisation.

2.1. Type de données du tableau

Les tableaux de variants peuvent contenir différents types de données, mais les valeurs de type `Variant` sont aussi celles qui prennent le plus de place en mémoire. Plus notre tableau sera grand, plus il occupera de la mémoire, d'autant plus si ses données sont des variants donc.

Ainsi, lorsque nous sommes sûrs que notre tableau ne va contenir qu'un seul type de données et qu'il va contenir beaucoup de valeurs, il est préférable d'indiquer ce type à la déclaration dans un souci d'optimisation.

```
' 1000 * 16 octets au minimum = 16 000 octets
Dim vTableau(1000) As Variant

' 1000 * 4 octets = 4000 soit 75% de moins pour cet exemple
Dim lTableau(1000) As Long
```

2.2. Dimensions

Jusqu'à présent, nous n'avons utilisé que des tableaux à une dimension. Or, il est possible de faire des tableaux contenant jusqu'à soixante dimensions ! Cela devient vite compliqué à se représenter 🍊 . Je ~~vous~~ nous rassure, nous nous contenterons de tableaux à une et deux dimensions dans la majeure partie des cas.

Pour rajouter des dimensions, il suffit d'ajouter des valeurs de limites pour chaque dimension :

```
Dim lTableau(limites lignes, limites colonnes, limites dimension
    3, ...) As Long
```

Comme indiqué dans ce morceau de pseudo-code, la première dimension correspond alors aux lignes, ce qui est un peu contre-intuitif par rapport aux tableaux à une dimension.

Nous accédons alors à une valeur en indiquant chacun des index constituant sa position :

```
valeur = tableau(index ligne, index colonne, index dimension3, ...)
```

(0, 0)	(0, 1)	...	(0, longueur dimension 2 -1)
(1, 0)	(1, 1)	...	(1, longueur dimension 2 -1)
(2, 0)	(2, 1)	...	(2, longueur dimension 2 -1)
(3, 0)	(3, 1)	...	(3, longueur dimension 2 -1)
...	...	...	...
(longueur dimension 1 - 1, 0)	(longueur dimension 1 - 1, 1)	...	(longueur dimension 1 - 1, longueur dimension 2 -1)

FIGURE 2.3. – Représentation tableau 2D (Option Base 0).

Voici un exemple plus complet :

```
1  Private Sub Exemple_TableauFixe_DeuxDimensions()
2      ' Tableau fixe de variant à 2 dimensions (2 lignes et 3
           colonnes)
3      Dim vTableauFixeDeuxDimensions(1, 2) As Variant
4
5      ' Première ligne
6      vTableauFixeDeuxDimensions(0, 0) = "Le Vieil Homme et la Mer"
7      vTableauFixeDeuxDimensions(0, 1) = "Ernest Hemingway"
8      vTableauFixeDeuxDimensions(0, 2) = 1952
9
10     ' Deuxième et dernière ligne
11     vTableauFixeDeuxDimensions(1, 0) = "Des souris et des hommes"
12     vTableauFixeDeuxDimensions(1, 1) = "John Steinbeck"
13     vTableauFixeDeuxDimensions(1, 2) = 1937
14
15     Sheets("Tableau fixe 2D").Range("A1:C2").Value =
           vTableauFixeDeuxDimensions
16 End Sub
```

	A	B	C	D
1	Le Vieil Homme et la Mer	Ernest Hemingway	1952	
2	Des souris et des hommes	John Steinbeck	1937	
3				
4				
5				
6				

FIGURE 2.4. – Résultat exemple tableau à deux dimensions.

2.3. Limites

Chaque dimension d'un tableau est constituée de limites de début et de fin. Il y a plusieurs choses à savoir les concernant.

2.3.1. Choix limites inférieure et supérieure

Jusqu'à présent, nous nous sommes contentés d'indiquer la limite supérieure pour chaque dimension (la limite inférieure étant automatiquement renseignée).

Néanmoins, il est possible de choisir également cette limite inférieure via l'usage de `To` dans les dimensions comme ceci :

```
Dim lTableau(limite inférieure lignes To limite supérieure lignes,
    limite inférieure colonnes To limite supérieure colonnes,
    limite inférieure dimension 3 To limite supérieure dimension
    3, ...)
```

Ce qui donnerait par exemple :

```
' Tableau fixe de deux lignes (index 4 et 5) et de trois colonnes
    (index 0, 1 et 2)
Dim lTableau(4 To 5, 2) As Long
```

2.3.2. Choix limite inférieure par défaut

Nous venons de dire que la limite inférieure était automatiquement renseignée. De base, elle est mise à 0, mais cela peut être changé via l'utilisation d'`Option Base` en début de fichier.

Cette instruction, à placer en tête de module, prend les valeurs 0 (par défaut) ou 1. Elle ne s'applique que pour les tableaux du fichier en cours.

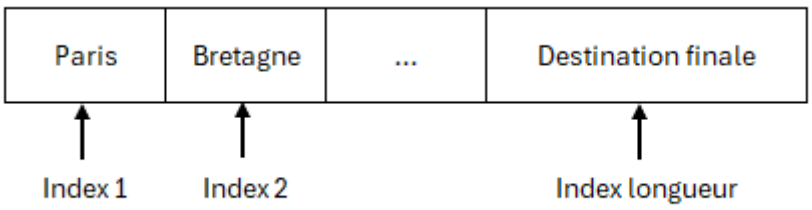

FIGURE 2.5. – Représentation tableau 1D (Option Base 1).

En vérité, il y a plusieurs cas particuliers qui font que peu importe l'utilisation de cette directive, le comportement ne sera jamais homogène, c'est pourquoi je laisse le comportement par défaut en n'utilisant pas cette instruction. Par exemple, les tableaux créés avec `Array` préfixé de `VBA.` ne seront pas impactés. Nous verrons d'autres cas particuliers plus loin dans ce billet.

```vba
' Instruction à placer tout en haut du fichier
' pour cet exemple uniquement /!\
Option Base 1 ' 0 par défaut

Private Sub Exemple_OptionBase_Avec_Array()
    Dim vTableau1() As Variant
    Dim vTableau2() As Variant

    vTableau1 = Array(1, 2, 3)
    vTableau2 = VBA.Array(1, 2, 3)
End Sub
```

Espions

Expression	Valeur	Type	Contexte
vTableau1		Variant/Variant(1 to 3)	Module1.Exemple_OptionBase_Avec_Array
vTableau1(1)	1	Variant/Integer	Module1.Exemple_OptionBase_Avec_Array
vTableau1(2)	2	Variant/Integer	Module1.Exemple_OptionBase_Avec_Array
vTableau1(3)	3	Variant/Integer	Module1.Exemple_OptionBase_Avec_Array
vTableau2		Variant/Variant(0 to 2)	Module1.Exemple_OptionBase_Avec_Array
vTableau2(0)	1	Variant/Integer	Module1.Exemple_OptionBase_Avec_Array
vTableau2(1)	2	Variant/Integer	Module1.Exemple_OptionBase_Avec_Array
vTableau2(2)	3	Variant/Integer	Module1.Exemple_OptionBase_Avec_Array

FIGURE 2.6. – Résultats tests Option Base et Array.

2.3.3. Récupération des limites

Comme nous l'avons évoqué, les tableaux ne commencent pas tous au même index et ne finissent pas tous au même index, selon le paramétrage et la façon dont le tableau a été fait.

Comment faire donc pour savoir où est le début et où la fin ?

Eh bien, c'est là qu'interviennent deux fonctions : `LBound` et `UBound`. Elles renvoient respectivement la valeur de la limite inférieure et la valeur de la limite supérieure, pour un tableau et une dimension donnés (première dimension par défaut).

```vba
' Tableau fixe de 9 lignes et 3 colonnes
Dim lTableau(2 To 10, 1 To 3) As Long
```

```
3
4  Debug.Print (LBound(lTableau, 1)) ' 2
5  Debug.Print (UBound(lTableau, 1)) ' 10
6  ' équivalent de
7  Debug.Print (LBound(lTableau)) ' 2
8  Debug.Print (UBound(lTableau)) ' 10
9
10 Debug.Print (LBound(lTableau, 2)) ' 1
11 Debug.Print (UBound(lTableau, 2)) ' 3
```

2.4. Redimensionnement

Jusqu'ici, nous avons pu utiliser des tableaux dynamiques qu'au travers la fonction `Array`, car nous n'avons pas encore vu le redimensionnement. Redimensionner un tableau consiste à modifier sa taille. L'instruction `ReDim` va nous servir à ça.

2.4.1. Sans préservation

Pour redimensionner sans préserver les valeurs, il suffit donc d'utiliser `ReDim` avec le tableau et l'expression des dimensions que nous avons déjà expliquée. Toutes les valeurs du tableau sont alors initialisées par défaut selon le type (0 pour les nombre, "" pour les chaînes de caractères ou encore vide pour les variants). Nous pouvons utiliser cette opération autant que voulue :

```
1  Private Sub Exemple_Redim_Sans_Preservation()
2      Dim iTableauDynamiqueEntier() As Integer '
3      ReDim iTableauDynamiqueEntier(1)
4      iTableauDynamiqueEntier(0) = 3
5      Debug.Print (iTableauDynamiqueEntier(0)) ' 3
6      Debug.Print (iTableauDynamiqueEntier(1)) ' 0
7
8      ReDim iTableauDynamiqueEntier(0 To 0, 1)
9      iTableauDynamiqueEntier(0, 0) = 4
10     Debug.Print (iTableauDynamiqueEntier(0, 0)) ' 4
11     Debug.Print (iTableauDynamiqueEntier(0, 1)) ' 0
12 End Sub
```

Notons aussi que le type peut être modifié, mais uniquement si le tableau est contenu dans une variable de type `Variant` :

```
1  Private Sub Exemple_Redim_Sans_Preservation_Type()
2      Dim vTableau As Variant
3      ReDim vTableau(3) As Long
4      ReDim vTableau(4) As String
5  End Sub
```

Sans préservation, toutes les valeurs sont réinitialisées.

2.4.2. Avec préservation

Pour pouvoir conserver les anciennes valeurs, il faut ajouter l'utilisation du mot-clef `Preserve` après `ReDim`. Le type du tableau n'est pas modifiable dans ce cas tout comme le nombre de dimensions une fois que défini. En fait, seule la dernière dimension est alors modifiable une fois le tableau dimensionné.

```
1  Private Sub Exemple_Redim_Avec_Preservation()
2      Dim iTableauDynamiqueEntier() As Integer '
3      ReDim iTableauDynamiqueEntier(1)
4      iTableauDynamiqueEntier(0) = 3
5      Debug.Print (iTableauDynamiqueEntier(0)) ' 3
6      Debug.Print (iTableauDynamiqueEntier(1)) ' 0
7
8      ReDim Preserve iTableauDynamiqueEntier(0)
9      Debug.Print (iTableauDynamiqueEntier(0)) ' 3
10 End Sub
```

Lorsque nous souhaitons préserver les valeurs lors d'un redimensionnement d'un tableau déjà dimensionné, seule la dernière dimension est modifiable. En conséquence, il arrive de devoir inverser la logique lignes/colonnes que nous avons vue concernant les tableaux 2D. En effet, dans certains programmes, il faut ajouter des lignes au fur et à mesure, donc réallouer de la place en mémoire au fur et à mesure. Ce ne serait pas possible sans placer les lignes en deuxième dimension dans ce cas. Nous pouvons nous dire que ça complique alors l'écriture de la plage si nécessaire. Certes, mais tant que ça, puisqu'il y a une fonction Excel de transposition dont nous parlerons un peu plus tard.

```
1  lNbLignes = 0
2  ReDim vLignes(lNbColonnes, lNbLignes )
3  lNbLignes = lNbLignes + 1
4  ' Seule la dernière dimension est modifiable avec Preserve
5  ReDim Preserve vLignes(lNbColonnes, lNbLignes )
```

2.5. Itération

Itérer sur un tableau consiste à le parcourir de sa limite inférieure à sa limite supérieure (ou inversement) pour chaque dimension :

```vba
Dim lIndexLigne As Long
Dim lIndexColonne As Long
For lIndexLigne = LBound(vTableauFixeDeuxDimensions) To
    UBound(vTableauFixeDeuxDimensions)
    For lIndexColonne = LBound(vTableauFixeDeuxDimensions, 2) To
        UBound(vTableauFixeDeuxDimensions, 2)
        ' ...
    Next lIndexColonne
Next lIndexLigne
```

Durant cette section, nous avons étudié des notions concernant le paramétrage de tableaux et leur parcours.

3. Autres façons de créer un tableau

Dans certains programmes, il est intéressant de créer des tableaux de manières plus spécifiques. C'est ce nous allons voir maintenant.

3.1. Affectation d'un tableau dynamique à partir…

Nous connaissons déjà la fonction `Array`, mais il y a encore différentes possibilités.

3.1.1. … d'une plage

Nous pouvons obtenir un tableau à partir d'une plage d'au minimum deux cellules par la propriété `Value` notamment.

Les tableaux lus ont alors tous deux dimensions (même si nous ne lisons qu'une ligne ou qu'une colonne) et sont bornés à partir de 1 indépendamment d'`Option Base`.

```vba
Private Sub Exemple_Lecture_Plages_Creation_Tableau()
    Dim vTableau1() As Variant
    vTableau1 = Sheets("Exemple plages").Range("A2:B30").Value
    Debug.Print LBound(vTableau1, 1), UBound(vTableau1, 1) ' 1 29
    Debug.Print LBound(vTableau1, 2), UBound(vTableau1, 2) ' 1 2

    Dim vTableau2() As Variant
    vTableau2 = Sheets("Exemple plages").Range("A32:A33").Value
    Debug.Print LBound(vTableau2, 1), UBound(vTableau2, 1) ' 1 2
    Debug.Print LBound(vTableau2, 2), UBound(vTableau2, 2) ' 1 1

    Dim vTableau3() As Variant
    vTableau3 = Sheets("Exemple plages").Range("A35:B35").Value
```

```
    Debug.Print LBound(vTableau3, 1), UBound(vTableau3, 1) ' 1 1
    Debug.Print LBound(vTableau3, 2), UBound(vTableau3, 2) ' 1 2
End Sub
```

3.1.2. … d'un autre tableau de chaînes de caractères

La fonction `Filter` de VBA permet de filtrer un tableau ne contenant que des chaînes de caractères. Pour ce faire, elle peut prendre jusqu'à quatre paramètres : le tableau, la valeur à trouver (ou non) dans chaque élément, un booléen pour indiquer si la sous-chaîne doit être trouvée (par défaut) ou non, ainsi qu'une valeur paramétrant le mode de comparaison.

Les tableaux retournés sont bornés à partir de 0 indépendamment d'`Option Base`.

```
Private Sub Exemple_Filter_Pour_Tableau()
    Dim vTableau() As Variant
    vTableau = Array("Paris", "Bretagne", "...",
        "Destination finale")

    Dim vTableauFiltre1 As Variant
    Dim vTableauFiltre2 As Variant
    Dim vTableauFiltre3 As Variant

    ' Tous les éléments
    vTableauFiltre1 = Filter(vTableau, "") ' "Paris", "Bretagne",
        "...", "Destination finale"

    ' Les éléments contenant 'e'
    vTableauFiltre2 = Filter(vTableau, "e") ' "Bretagne",
        "Destination finale"

    ' Les éléments ne contenant pas 'e'
    vTableauFiltre3 = Filter(vTableau, "e", False) ' "Paris", "..."
End Sub
```

Espions

Expression	Valeur	Type	Contexte
vTableau		Variant/Variant(0 to 3)	Module_Test.Exemple_Filter_Pour_Tableau
vTableau(0)	"Paris"	Variant/String	Module_Test.Exemple_Filter_Pour_Tableau
vTableau(1)	"Bretagne"	Variant/String	Module_Test.Exemple_Filter_Pour_Tableau
vTableau(2)	"..."	Variant/String	Module_Test.Exemple_Filter_Pour_Tableau
vTableau(3)	"Destination finale"	Variant/String	Module_Test.Exemple_Filter_Pour_Tableau
vTableauFiltre1		Variant/String(0 to 3)	Module_Test.Exemple_Filter_Pour_Tableau
vTableauFiltre1(0)	"Paris"	String	Module_Test.Exemple_Filter_Pour_Tableau
vTableauFiltre1(1)	"Bretagne"	String	Module_Test.Exemple_Filter_Pour_Tableau
vTableauFiltre1(2)	"..."	String	Module_Test.Exemple_Filter_Pour_Tableau
vTableauFiltre1(3)	"Destination finale"	String	Module_Test.Exemple_Filter_Pour_Tableau
vTableauFiltre2		Variant/String(0 to 1)	Module_Test.Exemple_Filter_Pour_Tableau
vTableauFiltre2(0)	"Bretagne"	String	Module_Test.Exemple_Filter_Pour_Tableau
vTableauFiltre2(1)	"Destination finale"	String	Module_Test.Exemple_Filter_Pour_Tableau
vTableauFiltre3		Variant/String(0 to 1)	Module_Test.Exemple_Filter_Pour_Tableau
vTableauFiltre3(0)	"Paris"	String	Module_Test.Exemple_Filter_Pour_Tableau
vTableauFiltre3(1)	"..."	String	Module_Test.Exemple_Filter_Pour_Tableau

FIGURE 3.7. – Résultat exemple fonction Filter.

3.1.3. … de la méthode d'évaluation d'Excel

Enfin, la méthode d'évaluation de l'application Excel permet aussi d'obtenir des tableaux.

Il faut alors faire appel à `Evaluate` ou bien utiliser le raccourci via la notation `[]` puis placer tous les éléments au sein d'accolades `{}` en les séparant par des virgules.

```
Dim vTableau() As Variant
' Version longue (Application pouvant être omis)
vTableau = Application.Evaluate(
    "{""Paris"", ""Bretagne"", ""..."", ""Destination finale""}")
' Version raccourcie avec les crochets
vTableau = [{"Paris", "Bretagne", "...", "Destination finale"}]
```

Les tableaux retournés sont bornés à partir de 1 indépendamment d'`Option Base`.

```
Private Sub Exemple_Evaluate_Pour_Tableau()
    Dim lIndex As Long
    Dim vTableau() As Variant
    ' Version longue (Application pouvant être omis)
    vTableau = Application.Evaluate(
        "{""Paris"", ""Bretagne"", ""..."", ""Destination finale""}"
        )
    ' Version raccourcie avec les crochets
    ' vTableau = [{"Paris", "Bretagne", "...", "Destination
       finale"}]

    Debug.Print (LBound(vTableau)) ' 1
    For lIndex = LBound(vTableau) To UBound(vTableau)
        Debug.Print (vTableau(lIndex))
        ' Paris
        ' Bretagne
        ' ...
        ' Destination finale
    Next lIndex
End Sub
```

i Cette approche est intéressante, car elle permet de créer des tableaux à deux dimensions également en utilisant alors un point-virgule pour séparer les lignes.

```vba
Dim vTableau() As Variant
vTableau = [{0, 1; 2, 3}]
' Ligne 1 : 0 1
' Ligne 2 : 2 3
```

3.2. Déclaration en tant que constante

Il n'est pas possible de déclarer un tableau comme une constante en VBA. Toutefois, nous pouvons utiliser des subterfuges pour pallier à cela, en utilisant une fonction retournant un tableau par exemple :

```vba
Public Function Exemple_Constante_Tableau()
    Exemple_Constante_Tableau = Array("Paris", "Bretagne", "...",
        "Destination finale")
End Function
```

Avec cet exemple, chaque appel à `Exemple_Constante_Tableau` nous donnera un tableau avec le même contenu :

```vba
Private Sub Exemple_Utilisation_Constante_Tableau()
    Dim vTableau() As Variant
    vTableau = Exemple_Constante_Tableau()

    vTableau(2) = "La Rochelle"
    Debug.Print (vTableau(2)) ' La Rochelle
    Debug.Print (Exemple_Constante_Tableau(2)) ' ...
End Sub
```

Cette section a présenté d'autres façons de créer des tableaux.

4. Opérations supplémentaires

Nous avons eu l'occasion de présenter certaines opérations tout au long de ces lignes. Nous allons en voir encore quelques unes.

4.1. Construire un texte à partir d'un tableau

Dans la section précédente, nous avons vu comment segmenter un texte selon un délimiteur afin de construire un tableau.

Ici, nous allons voir l'opération inverse, à savoir construire une chaîne de caractères à partir des valeurs d'un tableau et d'un délimiteur. Pour cela, nous ferons appel à la fonction native `Join` en passant respectivement ces deux arguments.

Cela peut être pratique pour fournir quelque chose de lisible à l'utilisateur par exemple :

```
1  Private Sub Exemple_Join_Tableau()
2      Dim sMessage As String
3      Dim bChoix As Boolean
4
5      sMessage = "Voulez-vous supprimer la/les ligne(s) : " &
           Join(Array(1, 3, 5), ", ") & " ?"
6
7      bChoix = MsgBox(sMessage, vbYesNo)
8
9      ' ...
10 End Sub
```

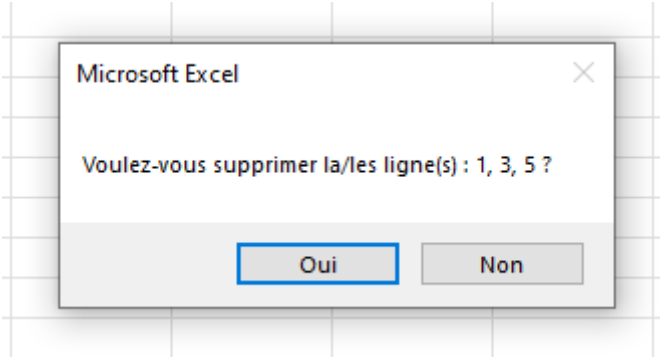

FIGURE 4.8. – Résultat exemple fonction Join.

4.2. Réinitialiser un tableau

Pour réinitialiser un tableau, nous utiliserons l'instruction `Erase` suivi du tableau. Son comportement diffère en fonction d'un tableau fixe et d'un tableau dynamique.

4.2.1. Tableau fixe

Appliquée à un tableau fixe, cette instruction réinitialise les valeurs du tableau à leurs valeurs par défaut selon le type du tableau (0 pour les entiers, vide pour les variants ou encore "" pour les chaînes de caractères par exemple).

```
1  Private Sub Exemple_Erase_TableauFixe()
2      Dim dTableauAEffacer(3) As Double
3      dTableauAEffacer(0) = 5
```

```
    Erase dTableauAEffacer
    Debug.Print (dTableauAEffacer(0)) ' 0
End Sub
```

4.2.2. Tableau dynamique

Appliquée à un tableau dynamique, cette instruction réinitialise complètement le tableau à un état vide, sans dimensions ni contenu.

```
Private Sub Exemple_Erase_TableauDynamique()
    Dim dTableauAEffacer() As Double
    ReDim dTableauAEffacer(3)
    dTableauAEffacer(0) = 5
    Erase dTableauAEffacer
    ' Erreur l'indice n'appartient pas à la sélection :
    ' Debug.Print (UBound(dTableauAEffacer))
End Sub
```

4.3. Transposer un tableau

Transposer un tableau consiste à transformer les lignes en colonnes et vice versa. Il y a une fonction Excel pour effectuer cette opération. Nous pouvons y accéder en VBA via `WorksheetFunction.Transpose`.

La transposition peut être une étape nécessaire pour écrire verticalement des tableaux à une dimension notamment.

```
Private Sub Exemple_Transposition_Tableau()
    ' Tableau fixe d'entier de 3 éléments
    Dim sDestinations(3) As String
    sDestinations(0) = "Paris"
    sDestinations(1) = "Bretagne"
    sDestinations(2) = "..."
    sDestinations(3) = "Destination finale"

    ' Tableau à 1 dimension s'écrit sur une ligne par défaut
    Sheets("Transposition").Range("A1:D1").Value = sDestinations

    ' Transposition du tableau en colonne
    Sheets("Transposition").Range("A1:A4").Value =
        WorksheetFunction.Transpose(sDestinations)
End Sub
```

	A	B	C	D	E
1	Paris	Bretagne	...	Destination finale	
2	Bretagne				
3	...				
4	Destination finale				
5					
6					

FIGURE 4.9. – Résultat exemple transposition.

4.4. Calculer le nombre d'éléments par dimension

Comme la limite inférieure d'une dimension ne commence pas toujours à 1, cela n'est pas toujours juste de se baser sur la limite supérieure pour déterminer le nombre d'éléments par dimension.

Voici une façon d'avoir à chaque fois le bon nombre d'éléments par dimension :

```
Nb éléments = UBound(Tableau, N° Dimension) - LBound(Tableau, N°
    Dimension) + 1
```

4.5. Lire et écrire des plages

Nous avons utilisé la lecture et l'écriture de plages dans certains exemples. Ajoutons deux choses.

Premièrement, recourir à des tableaux pour lire (puis éventuellement traiter) ou écrire des plages d'un coup plutôt que cellule par cellule est une bonne pratique d'un point de vue performances sur Excel. C'est d'ailleurs un point abordé dans un autre billet.

Deuxièmement, nous pouvons tirer profit de la méthode `Resize` pour renseigner la plage à écrire à partir d'une cellule de départ, en fournissant le nombre de lignes et de colonnes. C'est beaucoup plus simple et flexible que de devoir indiquer la plage soi-même.

```
Private Sub Exemple_LectureEtEcriture_Plages_Avec_Tableau()
    Dim vTableau() As Variant
    vTableau = Sheets("Exemple plages").Range("A2:B30").Value

    Sheets("Exemple plages").Range("D2").Resize( _
        UBound(vTableau, 1) - LBound(vTableau, 1) + 1, _
        UBound(vTableau, 2) - LBound(vTableau, 2) + 1 _
    ).Value = vTableau
End Sub
```

	A	B	C	D	E	F
1	A LIRE			A ECRIRE		
2	0	28		0	28	
3	1	27		1	27	
4	2	26		2	26	
5	3	25		3	25	
6	4	24		4	24	
7	5	23		5	23	
8	6	22		6	22	
9	7	21		7	21	
10	8	20		8	20	
11	9	19		9	19	
12	10	18		10	18	
13	11	17		11	17	
14	12	16		12	16	
15	13	15		13	15	
16	14	14		14	14	
17	15	13		15	13	
18	16	12		16	12	
19	17	11		17	11	
20	18	10		18	10	
21	19	9		19	9	
22	20	8		20	8	
23	21	7		21	7	
24	22	6		22	6	
25	23	5		23	5	
26	24	4		24	4	
27	25	3		25	3	
28	26	2		26	2	
29	27	1		27	1	
30	28	0		28	0	
31						

FIGURE 4.10. – Résultat exemple lecture et écriture de plages avec Resize.

Rappelons que les tableaux lus à partir des plages sont bornés à partir de 1 et non de 0, indépendamment d'`Option Base`, donc l'utilisation d'`UBound` dans l'exemple ci-dessus aurait aussi fonctionné.

Au fil de cette section, nous avons vu des opérations supplémentaires liées aux tableaux.

5. Nouveaux types et tableaux

Nous verrons l'ajout de types avec les structures et les classes dans un autre billet.

Les valeurs créées à partir d'une structure peuvent se placer dans un tableau ayant le type de cette structure.

```
' La structure TypeClient doit être déclarée auparavant dans le
    code
Dim tClient As TypeClient

Dim tcTableau1() As TypeClient
ReDim tcTableau1(0)
tcTableau1(0) = tClient
```

De même avec les classes :

```
' Le module de classe clsClient doit exister
Dim oClient As clsClient
Set oClient = New clsClient

Dim ocTableau() As clsClient
ReDim ocTableau(0)

Set ocTableau(0) = oClient
```

Toutefois, nous parlerons incessamment sous peu des collections qui peuvent être plus adaptées pour stocker ce type de valeurs.

Conclusion

C'est fini pour ce billet portant sur les tableaux. Nous y avons vu (ou revu) leur création et leur utilisation.

Bien que très pratiques, les tableaux ne sont pas toujours adaptés. Par exemple, il nous est impossible d'insérer facilement une nouvelle valeur à une position donnée (il nous faudrait alors redimensionner le tableau et décaler les valeurs nous-mêmes, ce qui est assez laborieux). Cela tombe bien, il existe d'autres structures de données telles que les collections qui fournissent une méthode pour cette opération. C'est d'ailleurs l'objet d'un autre billet !

À bientôt !

Quelques ressources :

— La documentation concernant la déclaration de tableaux
— La documentation concernant l'utilisation de tableaux
— La documentation concernant la fonction Array
— La documentation concernant l'instruction ReDim
— Cette page