Utiliser la classe Collection

Nous allons aborder l'utilisation des collections dans VBA Excel. Ce petit guide est inspiré du tutoriel de Excel Macro Mastery.

Ce guide est une présentation non exhaustive de l'utilisation des variables de type Collection en VBA, mais un débroussaillage pour vous donner une idée de ce qu'il est possible de faire avec ces variables, les avantages et les inconvénients.

Nous verrons comment déclarer une collection, comparer les collections avec des tableaux, leur utilisation avec des boucles et des objets.

La première chose à comprendre, c'est que lorsque nous parlons de la classe Collection en VBA Excel, nous devons garder en mémoire qu'il s'agit, un peu, comme d'une liste de valeur contenues dans une grosse variable. Avec ses avantages et ses inconvénients.

La classe Collection appartient à la librairie VBA.

Partie I – Les bases

Les membres de la classe Collection

La classe Collection nous offre 4 membres, 3 méthodes et 1 fonction.

- La méthode Add, pour ajouter un élément
- La méthode Remove, pour supprimer un élément
- La méthode Count, pour compter le nombre d'élément
- La fonction Item, pour accéder à un élément par son index

Nous étudierons, ensemble, l'utilisation de ces différents membres.

La déclaration et l'instanciation d'une Collection ?

Il y a deux manières de déclarer et d'instancier une collection, soit en une seule ligne, soit en deux lignes, mais le résultat final sera exactement le même, quel que soit la méthode utilisée.

```
Sub lesCollections()
    ' Première méthode
    Dim maColl As New Collection

    ' Seconde méthode
    Dim maColl2 As Collection

    Set maColl2 = New Collection

End Sub
```

Ces deux méthodes font précisément la même chose, elles instancient un objet de la classe Collection, alors pourquoi utiliser l'une ou l'autre méthode ?

La seconde méthode offre plus de flexibilité et nous permettrait de créer une collection au moment de son utilisation ou encore de vider une collection existante.

.Add - Ajouter des données à une collection

Pour ajouter un élément, nous allons utiliser la méthode .Add, dont voici la syntaxe.

Syntaxe : `objet.Add item, [Key], [ [After], [Before] ]`

`Item` est obligatoire, il représente la valeur à stocker et est de type Variant, indiquant que cette valeur peut être de n'importe quel type.

`Key` est optionnel et est, obligatoirement, de type chaîne de caractère.

`After` est optionnel et est de type entier ou chaîne de caractère, l'argument passé à After indique après quel membre ajouté l'élément.

`Before` est optionnel et est de type entier ou chaîne de caractère, l'argument passé à Before indique avant quel membre ajouté l'élément.

L'utilisation la plus simple pour instancier et ajouter un élément se résume à ces quelques lignes de codes VBA.

```
Sub lesCollections()
    Dim maColl As New Collection

    maColl.Add "Paul"

End Sub
```

Ce qui nous donnera comme résultat la création d'un objet `maColl`, dans lequel, nous aurons 1 membre contenant la valeur `"Paul"`.

Vous pouvez constater que le type de variable est Variant et qu'il est donc possible de stocker tous types de données, sans se soucier de leurs types.

A chaque ajout, la nouvelle valeur se positionnera à la fin de la collection.

```
Sub lesCollections()

    Dim maColl As New Collection

    maColl.Add "Paul"
    maColl.Add "Sophie"
    maColl.Add "Dominique"

End Sub
```

Ce qui aura pour effet de nous fournir le résultat suivant.

Nous pouvons décider d'ajouter un élément avant ou après un autre élément.

```
Sub lesCollections()

    Dim maColl As New Collection

    maColl.Add "Paul"
    maColl.Add "Sophie"
    maColl.Add "Dominique"
    maColl.Add "Nathalie", after:=1

End Sub
```

Ici, nous demandons d'ajouter la valeur Nathalie, après la première valeur.

De la même manière, nous pourrions demander à l'ajouter, avant un autre élément.

```
Sub lesCollections()

    Dim maColl As New Collection

    maColl.Add "Paul"
    maColl.Add "Sophie"
    maColl.Add "Dominique"
    maColl.Add "Nathalie", after:=1
    maColl.Add "Léon", before:=3

End Sub
```

Et le résultat sera alors le suivant.

Nous verrons un peu plus tard, qu'il est possible d'utiliser after et before avec des paramètres de type string.

.Count - Compter le nombre d'éléments d'une collection

Nous allons pouvoir utiliser la méthode .Count pour connaitre le nombre d'élément contenu dans notre collection.

Syntaxe : `objet.Count`

Cette méthode retourne une valeur de type entier long, contenant le nombre d'objets contenus dans la Collection. Cette valeur est en lecture seule.

```
Sub lesCollections()

    Dim maColl As New Collection

    maColl.Add "Paul"
    maColl.Add "Sophie"
    maColl.Add "Dominique"
    maColl.Add "Nathalie", after:=1
    maColl.Add "Léon", before:=3

    MsgBox "Nombre d'élément : " & maColl.Count

End Sub
```

Ces lignes nous donnerons l'apparition d'une boite de dialogue avec comme valeur 5.

Nous verrons un peu plus tard la différence entre une collection et un tableau, mais déjà, vous pouvez vous rendre compte que nous n'avons pas besoin de dimensionner notre collection, alors que nous devons le faire pour un tableau.

Mais également, si nous souhaitons insérer un nouvel élément dans un tableau, nous devons le redimensionner et recopier tous nos éléments vers le bas, avant d'insérer notre élément, ici, avec la Collection, Excel se charge de faire cela pour nous.

.Remove - Supprimer un élément

Syntaxe : `objet.Remove item`

`Item` est obligatoire, il représente la valeur que nous souhaitons supprimer.

Pour supprimer un élément de notre collection, nous allons utiliser la méthode .Remove et indiquer l'index de l'élément à supprimer, à noter que le premier élément possède l'index 1 et non 0.

```
Sub lesCollections()

    Dim maColl As New Collection

    maColl.Add "Paul"
    maColl.Add "Sophie"
    maColl.Add "Dominique"
    maColl.Add "Nathalie", after:=1
    maColl.Add "Léon", before:=3

    maColl.Remove 1

End Sub
```

Après avoir ajouté tous les éléments, nous supprimons le premier élément, nous allons alors comme résultat :

.Item – Accéder à un élément

Syntaxe : `objet.Item(item)`

`Item` est obligatoire, il représente la valeur à laquelle nous souhaitons accéder.

Nous pouvons donc très simplement accéder à un élément

```
Sub lesCollections()

    Dim maColl As New Collection

    maColl.Add "Paul", Key:="EL1"
    maColl.Add "Sophie", Key:="EL2"
    maColl.Add "Dominique", Key:="EL3"

    Debug.Print maColl.item(1)

End Sub
```

Ce code affichera bien sûr comme résultat : Paul

Veuillez noter qu'il est également possible de se passer de cette méthode en indiquant directement l'index à notre objet.

```
Sub lesCollections()

    Dim maColl As New Collection

    maColl.Add "Paul", Key:="EL1"
    maColl.Add "Sophie", Key:="EL2"
    maColl.Add "Dominique", Key:="EL3"

    Debug.Print maColl(1)

End Sub
```

Nous obtiendrons exactement le même résultat.

Option Key - Ajouter un élément avec une clé

En utilisant une clé, nous allons pouvoir utiliser notre Collection, comme un dictionnaire. Un dictionnaire est un ensemble de paires clé-valeur. Chaque valeur est associée à une clé unique qui sert d'identifiant.

Voici les caractéristiques d'un dictionnaire.

- Il garantit l'unicité de chaque élément en interdisant l'ajout de 2 clés identiques.
- Il permet d'accéder à un élément à l'aide sa clé
- Si la clé n'existe pas, nous aurons une erreur.
- Les clés sont obligatoirement de type chaîne de caractères.

Nous allons créer une collection de trois élèves que nous souhaitons ajouter avec une clé.

```
Sub lesCollections()

    Dim maColl As New Collection

    maColl.Add "Paul", Key:="EL1"
    maColl.Add "Sophie", Key:="EL2"
    maColl.Add "Dominique", Key:="EL3"

End Sub
```

Nous aurons comme résultat.

Comme vous pouvez le constater, nous ne voyons pas qu'il existe une clé et qu'elle est sa valeur, il sera donc très important de gérer les erreurs lors de la manipulation des clés, aussi bien en ajoutant une clé, pour gérer l'erreur de doublons ou l'appel à une clé qui n'existerait pas.

Pour afficher une valeur en appelant sa clé, nous pouvons utiliser ce code.

```
Sub lesCollections()

    Dim maColl As New Collection

    maColl.Add "Paul", Key:="EL1"
    maColl.Add "Sophie", Key:="EL2"
    maColl.Add "Dominique", Key:="EL3"

    Debug.Print maColl("EL2")

End Sub
```

Nous aurons alors comme résultat : Sophie

Il est également possible d'utiliser After et Before à l'aide d'une clé

```
Sub lesCollections()

    Dim maColl As New Collection

    maColl.Add "Paul", Key:="EL1"
    maColl.Add "Sophie", Key:="EL2"
    maColl.Add "Dominique", Key:="EL3"

    maColl.Add "Léon", Key:="EL4", before:="EL2"

End Sub
```

Et nous obtiendrons alors ce résultat

Partie II – Parcourir une collection

La boucle For Each

Nous allons pouvoir parcourir notre collection à l'aide de la boucle For Each.

```
Option Explicit

Sub lesCollections()
    Dim maColl As New Collection

    maColl.Add "Paul"
    maColl.Add "Sophie"
    maColl.Add "Dominique"
    maColl.Add "Nathalie", after:=1
    maColl.Add "Léon", before:=3

    Dim item As Variant
    For Each item In maColl
        Debug.Print item
    Next

End Sub
```

Après avoir déclaré notre collection, nous y ajoutons des valeurs et nous demandons ensuite que pour chaque élément de notre collection, nous souhaitons l'afficher dans notre fenêtre d'exécution. Pour le moment, ceci n'a aucun intérêt, mais cela nous permet de voir comment interagir avec notre collection.

Ce qui nous donnera

```
Exécution
Paul
Nathalie
Léon
Sophie
Dominique
```

La boucle For i

Une autre méthode, consiste à parcourir notre liste à l'aide d'un indice i.

```
Option Explicit

Sub lesCollections()
    Dim maColl As New Collection

    maColl.Add "Paul"
    maColl.Add "Sophie"
    maColl.Add "Dominique"
    maColl.Add "Nathalie", after:=1
    maColl.Add "Léon", before:=3

    Dim i As Long
    For i = 1 To maColl.Count
        Debug.Print maColl(i)
    Next i

End Sub
```

Avec pour résultat, la même chose que pour la boucle For Each.

Exécution

```
Paul
Nathalie
Léon
Sophie
Dominique
```

A noter que cette ligne est similaire à ce que nous venons de faire.

```
    For i = 1 To maColl.Count
        Debug.Print maColl.item(i)
    Next i
```

Nous verrons un peu plus tard la différence fondamentale entre ces deux boucles, mais pour le moment, une des choses qu'il est possible de faire facilement avec la seconde boucle est l'utilisation du paramètre step à notre boucle, pour lire un élément sur deux, par exemple.

```
Option Explicit

Sub lesCollections()
    Dim maColl As New Collection

    maColl.Add "Paul"
    maColl.Add "Sophie"
    maColl.Add "Dominique"
    maColl.Add "Nathalie", after:=1
    maColl.Add "Léon", before:=3

    Dim i As Long
    For i = 1 To maColl.Count Step 2
        Debug.Print maColl(i)
    Next i

End Sub
```

Et qui nous retournera

Exécution

```
Paul
Léon
Dominique
```

Cela n'aura probablement pas beaucoup d'intérêt, mais vous pourriez, cependant, parcourir la liste à l'envers.

```
Option Explicit

Sub lesCollections()
    Dim maColl As New Collection

    maColl.Add "Paul"
    maColl.Add "Sophie"
    maColl.Add "Dominique"
    maColl.Add "Nathalie", after:=1
    maColl.Add "Léon", before:=3

    Dim i As Long
    For i = maColl.Count To 1 Step -1
        Debug.Print maColl(i)
    Next i

End Sub
```

Et nous donnera comme résultat

Exécution

```
Dominique
Sophie
Léon
Nathalie
Paul
```

Mais nous pourrions également demander, les 5 premiers éléments ou les 5 derniers.

Choisir le type de boucle

Comme nous venons de le voir, la méthode For i permet plus de flexibilité pour lire notre collection, que si nous utilisons la boucle For Each.

Maintenant ce n'est pas le seul critère, un des critères les plus important est la différence de vitesse d'exécution de nos boucles.

Nous allons effectuer deux tests de simple affectation de variables, avec la boucle For Each et avec la boucle standard For.

Première boucle sur 50 000 éléments

```
For Each item In maColl
    s = item
Next item
```

La seconde boucle

```
For i = 1 To maColl.Count
    s = maColl(i)
Next i
```

La durée d'exécution de la boucle For Each est de 15 millisecondes

La durée d'exécution de la boucle For standard est de 10,3398 secondes, soit 661 fois plus lentes.

Ce qu'il faut en retenir c'est que la différence sera indétectable pour peu d'éléments, mais si le nombre d'éléments à traiter est élevé, vous devrez vraiment réfléchir à la boucle que vous voulez utiliser pour travailler avec votre collection et que l'utilisation de la boucle For Each sera probablement la plus appropriée.

Partie III – Travailler avec une feuille Excel

Nous allons commencer avec un tableau structuré, dans la feuille Feuil1 que nous avons nommé tblStock et qui contient ces données.

	A	B
1	Articles	Stock
2	Chaises	87
3	Bureau	5
4	Lampes	78
5	Tableau blanc	139
6	Tableau noir	295
7	Ramette papier A4	41
8	Gommes	24
9	Ordinateur portable	126
10	Ordinateur de bureau	275

Nous allons commencer par définir la plage de données à utiliser.

```
Option Explicit

Sub lireDonnees()
    Dim maColl As New Collection

    ' Obtenir la plage de données
    Dim plg As Range
    Set plg = Feuil1.Range("A1").CurrentRegion

End Sub
```

Ajouter des données à notre collection

Nous souhaitons maintenant, ajouter dans notre collection, les articles qui disposent de plus de 100 articles en stock.

```
Option Explicit

Sub lireDonnees()
    Dim maColl As New Collection

    ' Obtenir la plage de données
    Dim plg As Range
    Set plg = Feuil1.Range("A1").CurrentRegion

    Dim i As Long
    For i = 2 To plg.Rows.Count
        If plg.Cells(i, 2).Value > 100 Then
            maColl.Add plg.Cells(i, 1).Value
        End If
    Next i
End Sub
```

A savoir si nous devons écrire maColl.Add plg.Cells(i, 1).Value ou maColl.Add plg.Cells(i, 1)

Les deux possibilités sont offertes, mais ne retournent pas la même chose, la première méthode nous retournera directement la valeur, alors que la seconde méthode nous donnera un objet range.

Dans le cas de l'utilisation de .Value, nous aurons comme résultat

maColl		Collection/Collection
Item 1	"Tableau blanc"	Variant/String
Item 2	"Tableau noir"	Variant/String
Item 3	"Ordinateur portable"	Variant/String
Item 4	"Ordinateur de bureau"	Variant/String

Alors que sans l'utilisation de .Value, nous aurons

maColl	Collection/Collection
Item 1	Variant/Object/Range
Item 2	Variant/Object/Range
Item 3	Variant/Object/Range
Item 4	Variant/Object/Range

Tout dépendra de ce que vous souhaitez faire des données lues, mais vous pouvez voir qu'il est très facile d'ajouter des objets à notre collection, ce qui ouvre des portes considérables dans son utilisation.

Pour le moment, nous utiliserons .Value.

Envoyer une collection à une procédure.

Nous voulons afficher le résultat de notre traitement et pour cela, nous allons créer une procédure.

```
Sub afficheCollection(coll As Collection)
    Dim element As Variant
    For Each element In coll
        Debug.Print element
    Next element
End Sub
```

Et nous allons pouvoir appeler cette procédure en passant en paramètre la collection à afficher.

```
Option Explicit

Sub lireDonnees()
    Dim maColl As New Collection

    ' Obtenir la plage de données
    Dim plg As Range
    Set plg = Feuil1.Range("A1").CurrentRegion

    Dim i As Long
    For i = 2 To plg.Rows.Count
        If plg.Cells(i, 2).Value > 100 Then
            maColl.Add plg.Cells(i, 1).Value
        End If
    Next i
    Call afficheCollection(maColl)
End Sub
```

Ce qui nous donnera comme résultat

Exécution

```
Tableau blanc
Tableau noir
Ordinateur portable
Ordinateur de bureau
```

Ecrire le résultat dans une feuille

Nous souhaitons maintenant écrire le résultat dans notre feuille Excel, il existe de multiples possibilités pour le faire, mais nous allons le faire le plus simplement possible.

Nous souhaitons écrire nos données à partir de la cellule E1, puis E2, etc…

Nous allons écrire une procédure pour cela.

```
Sub ecrireCollection(coll As Collection)
    Dim element As Variant, ligne As Long
    ligne = 1
    For Each element In coll
        Feuill.Cells(ligne, 5).Value = element
        ligne = ligne + 1
    Next element
End Sub
```

Nous n'avons plus qu'à appeler cette procédure, comme nous l'avions fait pour afficher le résultat dans notre fenêtre d'exécution.

```
    Dim i As Long
    For i = 2 To plg.Rows.Count
        If plg.Cells(i, 2).Value > 100 Then
            maColl.Add plg.Cells(i, 1).Value
        End If
    Next i
    Call ecrireCollection(maColl)
End Sub
```

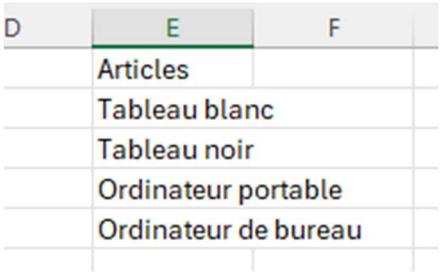
Et nous obtenons bien le résultat attendu dans notre feuille de calcul Excel.

D	E	F
	Tableau blanc	
	Tableau noir	
	Ordinateur portable	
	Ordinateur de bureau	

Si nous souhaitons inclure l'en-tête dans notre écriture, il nous suffit de très légèrement modifier notre procédure lireDonnees().

```
Dim i As Long
For i = 1 To plg.Rows.Count
    If plg.Cells(i, 2).Value > 100 Or i = 1 Then
        maColl.Add plg.Cells(i, 1).Value
    End If
Next i
```

Nous commençons notre boucle à 1 et nous ajoutons le test logique pour inclure cette ligne d'en-tête à notre collection.



La question que vous pouvez vous poser maintenant, c'est, mais si je veux avoir plus d'une colonne dans ma collection, comment je peux faire ?

Nous répondrons à cette question un peu plus tard avec l'utilisation des classes, mais sachez déjà, que c'est tout à fait possible. Mais voyons déjà quelle est la différence entre une collection et un tableau de type Array.

Partie IV – Différence entre Collection et Array

Nous allons voir dans cette partie, la différence qu'il peut-y avoir entre l'utilisation d'une collection et d'un tableau, en VBA, dans différentes situations.

Prenons comme exemple les données contenues dans la feuille Feuil2, dans le tableau tblRecensement, contenant 5 colonnes et 195 enregistrements.

	A	B	C	D	E
1	Code arrondissement	Code canton	Code commune	Nom de la commune	Population totale
2	1	08	001	Abbéville-la-Rivière	340
3	1	08	016	Angerville	4 449
4	3	05	017	Angervilliers	1 755
5	3	01	021	Arpajon	11 382
6	1	08	022	Arrancourt	127
7	3	02	027	Athis-Mons	36 527
8	1	08	035	Authon-la-Plaine	380
9	2	13	037	Auvernaux	int
10	1	08	038	Auvers-Saint-Georges	1 297
11	3	01	041	Avrainville	1 038
12	3	11	044	Ballainvilliers	4 797
13	2	13	045	Ballancourt-sur-Essonne	7 868
14	1	13	047	Baulne	1 431
15	3	10	064	Bièvres	4 818
16	1	08	067	Blandy	115

Lire l'ensemble d'un tableau

Avec une collection

Nous avons déjà écrit le code pour récupérer les données dans notre collection, ici, la population totale par commune du département 91.

```
Option Explicit

Sub lireDonnees()
    Dim maColl As New Collection

    ' Obtenir la plage de données
    Dim plg As Range
    Set plg = Feuil2.Range("A1").CurrentRegion

    Dim i As Long
    For i = 1 To plg.Rows.Count
        maColl.Add plg.Cells(i, 5).Value
    Next i

End Sub
```

Comme nous l'avons expliqué, c'est un des inconvénients des collections, nous devons lire chaque ligne de notre tableau structuré contenu dans notre feuille puis ajouter les valeurs, les unes après les autres.

L'autre inconvénient est que nous ne pouvons ajouter dans notre collection qu'une valeur, nous ne pouvons pas ajouter, par exemple, le code de la commune, son nom et le nombre d'habitant, nous verrons comment faire avec une classe.

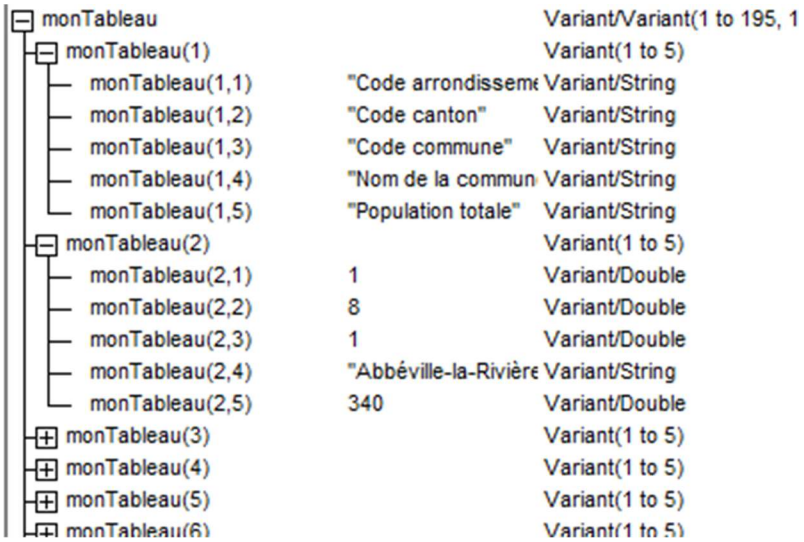
Avec une variable Array

Ecrivons le code pour récupérer les données dans notre variable

```
Sub lireDonneesTableau()

    ' Obtenir la plage de données
    Dim monTableau As Variant
    monTableau = Feuil2.Range("A1").CurrentRegion.Value

End Sub
```

Et c'est tout ce que nous avons besoin d'écrire, pas besoin de boucle, de dimensionner le tableau ou d'autres choses.

Et nous obtenons



Et si nous souhaitons écrire les données contenues dans notre variable Array dans une feuille de calcul Excel, c'est aussi simple.

```
Sub lireDonneesTableau()

    ' Obtenir la plage de données
    Dim monTableau As Variant
    monTableau = Feuil2.Range("A1").CurrentRegion.Value

    ' Écrire les données dans une feuille
    Feuil2.Range("G1:K200").Value = monTableau

End Sub
```

Quand nous avons besoin de copier tout un tableau de données structurées, la variable de type Array est beaucoup plus rapide et efficace que la Collection.

Filtrer des données

Avec une collection

Nous souhaitons, cette fois-ci, récupérer les données, uniquement du 11ème canton.

	A	B	C	D	E
1	Code arrondissement	Code canton	Code commune	Nom de la commune	Population totale
11	3	11	044	Ballainvilliers	4 797
44	3	11	136	Champlan	2 619
65	3	11	216	Épinay-sur-Orge	10 870
92	3	11	665	La Ville-du-Bois	8 175
104	3	11	339	Linas	7 082
106	3	11	345	Longjumeau	20 750
122	3	11	425	Montlhéry	8 897
166	3	11	587	Saulx-les-Chartreux	6 639

```
Sub lireDonnees()
    Dim maColl As New Collection

    ' Obtenir la plage de données
    Dim plg As Range
    Set plg = Feuil2.Range("A1").CurrentRegion

    Dim i As Long
    For i = 1 To plg.Rows.Count
        If plg.Cells(i, 2).Value = 11 Then
            maColl.Add plg.Cells(i, 5).Value
        End If
    Next i

End Sub
```

Avec la collection, nous ajoutons un test logique et n'ajoutons que les valeurs appartenant au 11ème canton.

Avec une variable Array

Cette fois-ci, le code devient un peu plus long.

```
Sub lireDonneesTableau()

    ' Déclaration du tableau recevant les données
    Dim monTableau() As Long

    ' Obtenir la plage de données
    Dim plg As Range
    Set plg = Feuil2.Range("A1").CurrentRegion

    ' Définir les variables utiles pour la boucle
    Dim i As Long, ligne As Long
    ligne = 1
    For i = 1 To plg.Rows.Count
        If plg.Cells(i, 2).Value = 11 Then
            ReDim Preserve monTableau(1 To ligne)
            monTableau(ligne) = plg.Cells(i, 5).Value
            ligne = ligne + 1
        End If
    Next i

End Sub
```

Nous avons dû ajouter du code pour gérer notre tableau, une variable ligne, une initialisation de cette variable, le redimensionnement de notre tableau et l'incrément de la ligne.

Dans ce cas de besoin, la collection est plus simple à utiliser.

Ajouter des données en les positionnant

Avec une collection

Un troisième cas de figure, qui montrera la différence entre une collection et une variable de type Array, nous souhaitons ajouter des données en mode LIFO (Last In First Out), cela signifie que nous remplissons notre tableau et que nous souhaitons que le dernier élément entré soit le premier de la liste.

```
Sub insertionLIFO()
    Dim maColl As New Collection

    maColl.Add "Pomme"
    maColl.Add "Poire", before:=1
    maColl.Add "Fraise", before:=1
    maColl.Add "Orange", before:=1
    maColl.Add "Citron", before:=1
End Sub
```

Donnant comme résultat

Nous pouvons constater que cela fonction très bien et que nous obtenons le résultat souhaité très facilement.

Avec une variable de type Array

Nous commençons par écrire la procédure principale.

```
Sub insertLIFOTab()
    Dim monTableau() As String

    ReDim monTableau(1 To 1)
    monTableau(1) = "Pomme"
    monTableau = fctTablInsert(monTableau, "Poire", 1)
    monTableau = fctTablInsert(monTableau, "Fraise", 1)
    monTableau = fctTablInsert(monTableau, "Orange", 1)
    monTableau = fctTablInsert(monTableau, "Citron", 1)
End Sub
```

Nous avons écrit une fonction pour insérer un élément à une position données.

Voyons le code de cette fonction fctTablInsert()

```
Function fctTablInsert(monTableau() As String, valeur As String, pos As Long) As String()
    ' Ajout un à la taille du tableau
    ReDim Preserve monTableau(LBound(monTableau) To UBound(monTableau) + 1)

    ' Déplacement des valeurs vers la nouvelle position
    Dim i As Long
    For i = UBound(monTableau) To pos + 1 Step -1
        monTableau(i) = monTableau(i - 1)
    Next i

    ' Ajout de la valeur à la position données
    monTableau(pos) = valeur

    ' retourner le tableau
    fctTablInsert = monTableau
End Function
```

Nous obtenons

Nous avons le même résultat que pour la collection, mais vous comprenez bien que c'est beaucoup plus difficile à mettre en place et bien plus long.

Dans ce cas-là, c'est-à-dire, lorsque nous souhaitons insérer un élément à une position données dans notre variable, l'utilisation de la Collection est indéniablement plus adaptée.

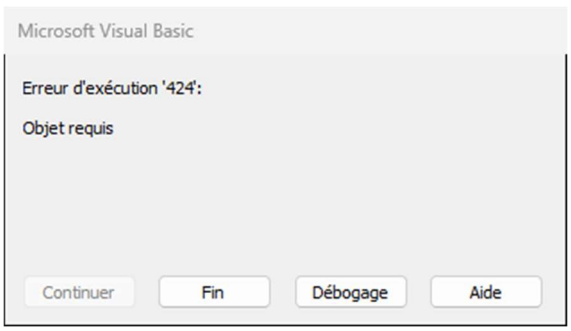
Modifier une valeur

Avec une collection

Nous avons une collection de données et nous souhaitons modifier une valeur.

```
Sub modifValeur()
    Dim maColl As New Collection

    maColl.Add "Pomme"
    maColl.Add "Poire"
    maColl.Add "Fraise"

    maColl(1) = "Asperge"
End Sub
```

Si nous exécutons ce code, nous avons un message d'erreur



L'élément d'une collection est en lecture seule, il n'est donc pas possible de modifier la valeur, sauf si nous utilisons une classe, mais pour le moment, avec les types de bases, ce n'est pas possible de cette manière. Nous devons d'abord supprimer la valeur souhaitée, puis ajouter une nouvelle valeur, à sa position.

```
Sub modifValeur()
    Dim maColl As New Collection

    maColl.Add "Pomme"
    maColl.Add "Poire"
    maColl.Add "Fraise"

    maColl.Remove 1
    maColl.Add "Asperge", before:=1

End Sub
```

Avec une variable Array

Nous souhaitons faire la même chose, mais avec un tableau, nous écrivons donc le code suivant.

```
Sub modifValeur()
    Dim monTableau() As String

    ReDim monTableau(1 To 3)
    monTableau(1) = "Pomme"
    monTableau(2) = "Poire"
    monTableau(3) = "Fraise"

    monTableau(1) = "Asperge"
End Sub
```

Cette fois-ci, nous n'avons aucun problème. Dans ce cas-là, l'utilisation d'un tableau est un peu plus simple qu'avec une variable de type Collection.

En résumé

Une variable de type Array

- Facile pour lire l'ensemble des données d'un tableau structuré dans une feuille
- Modification facile d'un élément du tableau
- Nous devons connaitre la taille du tableau ou redimensionner le tableau en permanence.
- Il n'est pas aisé d'insérer un élément ou de le supprimer.

Une variable de type Collection

- Plus de traitement pour lire l'ensemble des données d'un tableau
- Plus aisé pour lire des données à filtrer
- Impossible de mettre à jour une donnée (Sauf avec un objet)
- Inutile de connaitre la taille à l'avance et de redimensionner la variable
- Facilité pour insérer ou supprimer un élément

Partie V – Utilisation d'une collection avec une classe

Les données que nous allons utiliser.

	A	B	C	D	E
1	Code arrondissement	Code canton	Code commune	Nom de la commune	Population totale
2	1	08	001	Abbéville-la-Rivière	340
3	1	08	016	Angerville	4 449
4	3	05	017	Angervilliers	1 755
5	3	01	021	Arpajon	11 382
6	1	08	022	Arrancourt	127
7	3	02	027	Athis-Mons	36 527
8	1	08	035	Authon-la-Plaine	380
9	1	08	038	Auvers-Saint-Georges	1 297
10	3	01	041	Avrainville	1 038
11	3	11	044	Ballainvilliers	4 797
12	2	13	045	Ballancourt-sur-Essonne	7 868
13	1	13	047	Baulne	1 431

Nous avons déjà vu comment lire des données d'une feuille Excel et de placer une valeur dans une variable de type Collection, nous avons également vu qu'il n'était possible d'y positionner qu'un seul élément, sauf à utiliser des classes et c'est ce que nous allons voir dans cette partie.

Commençons par écrire notre procédure de lecture de données.

```
Sub lectureDonnees()
    Dim maColl As New Collection

    Dim plg As Range
    Set plg = Feuil2.Range("A1").CurrentRegion

    Dim i As Long
    For i = 2 To plg.Rows.Count
        If plg.Cells(i, 2).Value = 11 Then
            maColl.Add plg.Cells(i, 5).Value
        End If
    Next i
End Sub
```

Nous lisons l'ensemble des données et pour chaque champ égal à 11, nous ajoutons une valeur dans notre collection.

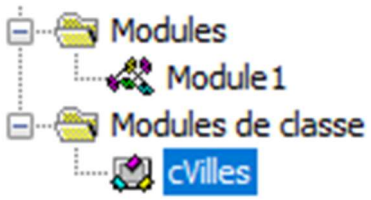
Maintenant, ce que nous souhaitons, c'est ajouter l'ensemble des valeurs contenues dans l'enregistrement (une ligne du tableau).

Nous allons donc créer une classe.

Création d'une classe

La classe que nous allons créer sera particulièrement simple, et nous devrions plutôt opter pour l'écriture d'une classe sérieuse, avec des Getters, des Setters, des

méthodes et des fonctions, mais comme nous travaillons sur les Collections, nous ne nous attarderons pas sur l'écriture d'une classe. Commençons par ajouter un module de classe et de le nommer, nous, nous lui avons donné le nom cVilles.



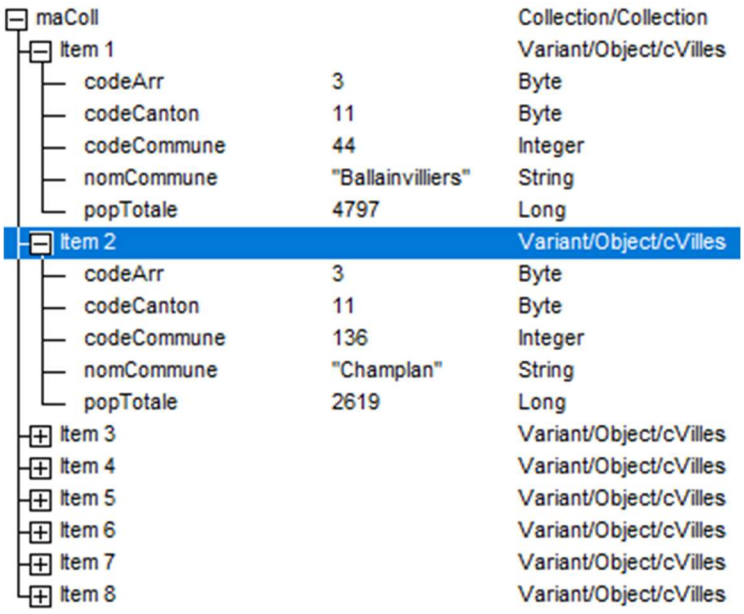

```
' Class cVilles
Public codeArr As Byte
Public codeCanton As Byte
Public codeCommune As Integer
Public nomCommune As String
Public popTotale As Long
```

Utilisation de la classe

Nous n'avons plus, maintenant, qu'à utiliser notre classe pour chaque enregistrement.

```
Sub lectureDonnees()
    Dim maColl As New Collection

    Dim plg As Range
    Set plg = Feuil2.Range("A1").CurrentRegion

    Dim ville As cVilles

    Dim i As Long
    For i = 2 To plg.Rows.Count
        If plg.Cells(i, 2).Value = 11 Then
            Set ville = New cVilles

            ville.codeArr = plg.Cells(i, 1).Value
            ville.codeCanton = plg.Cells(i, 2).Value
            ville.codeCommune = plg.Cells(i, 3).Value
            ville.nomCommune = plg.Cells(i, 4).Value
            ville.popTotale = plg.Cells(i, 5).Value

            maColl.Add ville
        End If
    Next i
End Sub
```

Et en exécutant notre code, nous obtenons notre collection complétée.



Exemple

Nous souhaitons connaitre le nombre total de la population pour notre sélection, nous allons donc utiliser notre collection.

```
Dim enrOut As cVilles
Dim totalVille As Long

totalVille = 0
For i = 1 To maColl.Count
    Set enrOut = maColl(i)
    totalVille = totalVille + enrOut.popTotale
Next i
Debug.Print "Population totale : " & totalVille
```

Et nous obtenons alors

```
Exécution
Population totale : 69829
```