Vk

Zeste de savoir

Les collections et les dictionnaires en VBA

28 avril 2023

Table des matières

Introduction

En informatique, les structures de données sont, comme leur nom l'indique, un moyen d'organiser les données. C'est l'usage voulu, avec la nature des données et des opérations à effectuer, qui permettra de définir la structure de données la plus adaptée.

Le langage de macro VBA apporte dans son sillage des structures de données. Vous connaissez peut-être les tableaux (`Array`).

Au cours de ce billet, nous allons en étudier deux en particulier: les collections et les dictionnaires.

> *i* Pour ce billet, nous nous placerons dans l'environnement Microsoft Office.

C'est parti!

1. Les collections

Une collection permet de contenir des éléments de façon ordonnée. Ses membres ne doivent pas nécessairement avoir le même type.

Les collections sont très répandues dans l'interface de programmation de Microsoft Office et vous avez sans doute l'habitude de les utiliser au quotidien! La propriété `Worksheets` représentant les feuilles d'un classeur est une collection par exemple.

1.1. Créer une collection

Nous pouvons déclarer puis instancier un objet de type `Collection` de la sorte:

```
1 Dim cNotes As Collection
2 Set cNotes = New Collection
```

1.2. Ajouter des éléments à une collection

La méthode `Add` permet d'ajouter des éléments à une collection. En second argument, il est possible d'indiquer une clef sous forme de chaîne de caractères, sorte d'alias qui nous permettra de manipuler l'élément au même titre que son index dans les méthodes.

```
1 cNotes.Add 18, "top"
2 cNotes.Add (12)
3 cNotes.Add (14)
```

1.3. Supprimer des éléments d'une collection

La méthode `Remove` permet de supprimer des éléments d'une collection. Nous pouvons lui passer la position ou la clef de l'élément à enlever.

```
1 cNotes.Remove (1) 'équivalent de cNotes.Remove ("top")
```

Lorsqu'un élément est supprimé, les autres sont réordonnés en conséquence.

1.4. Accéder à un élément d'une collection

Pour accéder à un élément en fonction de son index ou de sa clef, il y existe la méthode `Item` ou encore le sucre syntaxique `()`.

```
1 Debug.Print (cNotes.Item(1)) ' 12
2 Debug.Print (cNotes(1)) ' 12
```

1.5. Obtenir la longueur d'une collection

La propriété `Count` nous permet de savoir le nombre d'éléments présents dans la collection.

```
1 Debug.Print (cNotes.Count) ' 2
```

1.6. Itérer sur une collection

Les collections peuvent être parcourues avec une boucle `For Each`.

```
1 Dim vNote As Variant
2 For Each vNote In cNotes
3     Debug.Print (vNote)
4     ' 12
5     ' 14
6 Next vNote
```

Il est tout à fait possible d'itérer avec une boucle `For` en connaissant la longueur de la collection avec les index de la même façon que sur un tableau.

1.7. Modifier un élément dans une collection

Aucune méthode ne nous permet de modifier un élément directement. L'alternative est de supprimer l'élément de la collection et d'insérer le nouveau à la bonne place en indiquant un paramètre `Before` ou `After` à la méthode `Add`.

```vba
'cNotes(1) = 20 ' Erreur 424
cNotes.Remove (1)
cNotes.Add 20, Before:=1
```

1.8. Vider une collection

Il n'existe pas de méthode toute prête pour vider une collection. Une possibilité est de boucler sur tous les index de la fin au début et de supprimer chaque élément individuellement.

Une autre possibilité est de réinitialiser la collection.

```vba
Set cNotes = Nothing
Set cNotes = New Collection
Debug.Print (cNotes.Count) ' 0
```

Au cours de cette section, nous avons vu ce qu'étaient les collections et comment s'en servir en VBA.

2. Les dictionnaires

Dans la section précédente, nous avons vu que les collections fonctionnaient avec un principe de paire index/valeur, mais que nous pouvions donner une chaîne de caractères en clef pour aussi fonctionner avec une paire clef/valeur.

Ce dernier comportement rejoint celui des dictionnaires qui sont des tables dont les clefs sont des chaînes de caractères. Les dictionnaires sont plus adaptés à cet usage.

Avant de commencer, nous allons ajouter la référence *Microsoft Scripting Runtime* comme illustré ci-dessous:

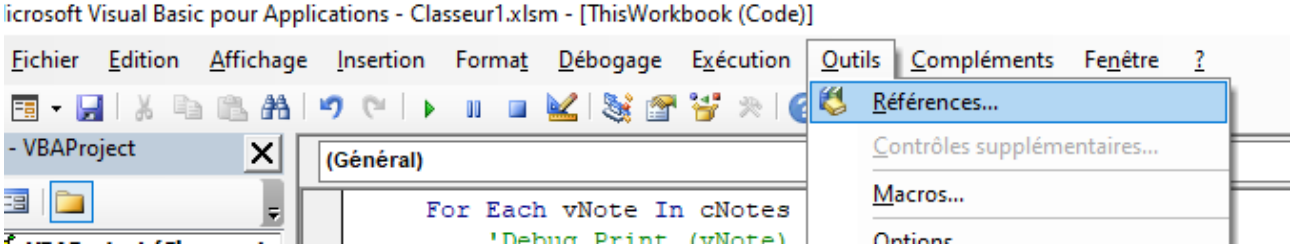

FIGURE 2.1. – Accès à la fenêtre des Références

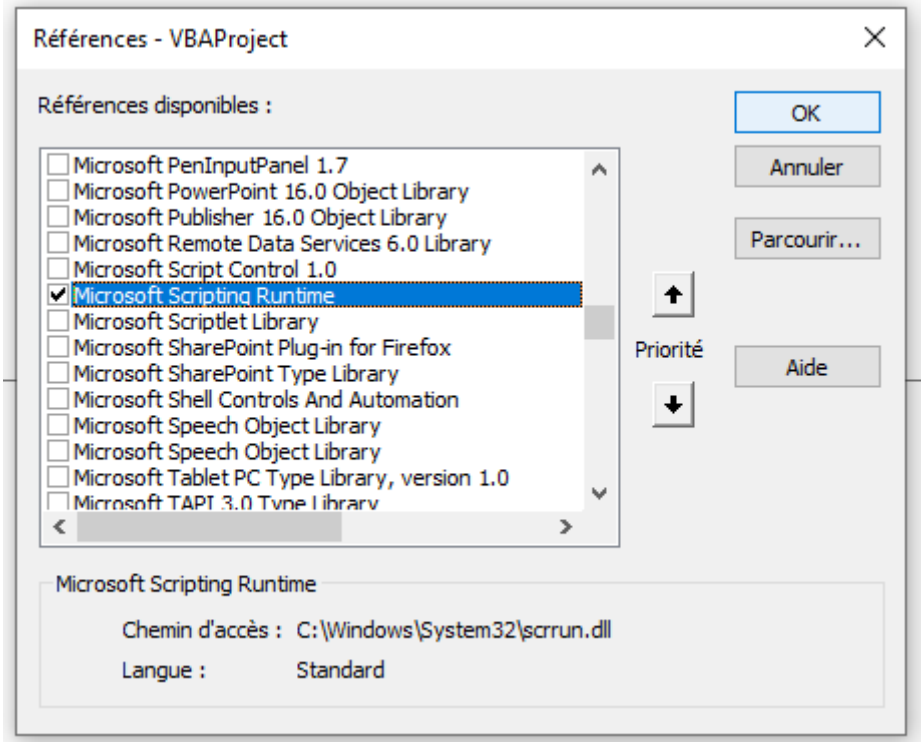

FIGURE 2.2. – Ajout référence Microsoft Scripting Runtime

2.1. Créer un dictionnaire

Nous pouvons déclarer puis instancier un objet dictionnaire de la sorte:

```
1 Dim dictFruits As Scripting.Dictionary
2 Set dictFruits = New Scripting.Dictionary
```

2.2. Ajouter des éléments à un dictionnaire

Pour ajouter des éléments à un dictionnaire, nous pouvons utiliser la méthode `Add` en indiquant une clef en premier argument et un élément en second.

```
1 dictFruits.Add "pomme", 3
2 dictFruits.Add "framboise", 7
3 dictFruits.Add "kiwi", 2
```

i Comme les collections, les dictionnaires peuvent contenir des éléments de types différents.

2.3. Supprimer des éléments d'un dictionnaire

Nous pouvons supprimer un élément au travers la méthode `Remove` en indiquant la clef de l'élément à supprimer.

```
dictFruits.Remove ("framboise")
```

2.4. Accéder à un élément d'un dictionnaire

Pour accéder à un élément en fonction de sa clef, il y existe la méthode `Item` ou encore le sucre syntaxique `()`.

```
Debug.Print (dictFruits.Item("pomme")) ' 3
Debug.Print (dictFruits("pomme")) ' 3
```

2.5. Obtenir la longueur d'un dictionnaire

La propriété `Count` nous permet de savoir le nombre d'éléments présents dans le dictionnaire.

```
Debug.Print (dictFruits.Count) ' 2
```

2.6. Vérifier l'existence d'une clef dans le dictionnaire

Nous pouvons vérifier si une clef est présente ou non dans un dictionnaire avec la méthode `Exists` en indiquant la clef à tester.

```
If dictFruits.Exists("framboise") Then
    Debug.Print ("framboise dans dictFruits")
Else
    Debug.Print ("pas de framboise")
End If ' pas de framboise
```

2.7. Itérer sur un dictionnaire

La propriété `Items` contient toutes les valeurs et nous permet d'itérer de la sorte:

```
Dim vPrix As Variant
For Each vPrix In dictFruits.Items
    Debug.Print (vPrix)
    ' 3
    ' 2
Next vPrix
```

La propriété `Keys` contient toutes les clefs et nous permet d'itérer en connaissant les clefs:

```
Dim vFruit As Variant
For Each vFruit In dictFruits.Keys
    Debug.Print vFruit, dictFruits(vFruit)
    ' pomme 3
    ' kiwi 2
Next vFruit
```

2.8. Modifier une valeur dans un dictionnaire

Contrairement aux collections, il est possible de modifier un élément de dictionnaire. Cela se fait ainsi:

```
dictFruits("pomme") = 5
Debug.Print (dictFruits("pomme")) ' 5
```

2.9. Modifier une clef dans un dictionnaire

Nous pouvons aussi modifier une clef pour la renommer par exemple.

```
dictFruits.Key("pomme") = "pommeverte"
Debug.Print (dictFruits.Exists("pomme")) ' Faux
```

2.10. Vider un dictionnaire

Nous pouvons supprimer l'intégralité du contenu d'un dictionnaire à l'aide de sa méthode `RemoveAll`.

```
dictFruits.RemoveAll
Debug.Print (dictFruits.Count) ' 0
```

Durant cette section, nous avons vu ce qu'étaient les dictionnaires et comment les utiliser en VBA.

Conclusion

C'est déjà la fin de ce billet.

Au cours de celui-ci, nous avons découvert deux structures de données utilisables en VBA: les collections et les dictionnaires. Celles-ci s'utilisent en fonction des besoins pour nos macros.

Certaines structures ne sont pas disponibles de base et doivent être implémentées soi-même comme les arbres par exemple.

À bientôt!

Quelques ressources:

— La documentation concernant les collections
— La documentation concernant les dictionnaires

Liste des abréviations

VBA Visual Basic for Applications. 1