

Excel-VBA

Cours intégral

Issa KACHAOU

Délégué étudiant du M1 MBFA parcours Ingénierie Immobilière

UNIVERSITÉ PARIS EST-CRÉTEIL
Département d'économie
2024

Table des matières

1	Les Bases des Macros et des Fonctions	6
1.1	Introduction à VBA	6
1.2	Structure d'un Code VBA	6
1.2.1	Les Macros (Subroutines)	6
1.2.2	Les Fonctions	7
1.2.3	Différence entre Sub et Function	8
1.3	Comment Utiliser les Macros et les Fonctions	8
1.3.1	Exécution d'une macro	8
1.3.2	Utilisation d'une fonction dans une cellule Excel	8
1.4	Résumé	9
1.5	Conclusion	9
2	Utilisation de la fonction MsgBox en VBA	10
2.1	Exemple simple de MsgBox	10
2.2	Personnalisation de MsgBox	10
2.3	Retour à la ligne dans MsgBox	11
2.4	Confirmation avec MsgBox	11
2.5	Exemple avancé : Mise en couleur automatique	12
3	Gestion des erreurs et affichage des informations d'un pays	13
3.1	Théorie : Les notions abordées	13
3.1.1	La gestion des erreurs	13
3.1.2	Les interactions utilisateur	14
3.1.3	La gestion des plages de données	14
3.2	Exemples pratiques avec explications	15
3.2.1	Exemple 1 : Gestion des erreurs avec TypeVal	15
3.2.2	Exemple 2 : Affichage des informations d'un pays	16
3.3	Conclusion	17

4	Introduction aux Variables en VBA	18
4.1	Déclaration des Variables et Portée	18
4.2	Variables à Longueur Fixe	19
5	Les Boucles For et While	20
5.1	Introduction	20
5.2	La Boucle For	20
5.3	La Boucle Do While	21
5.4	Comparaison entre For et Do While	22
5.5	Exemple Combiné : For et Do While	23
5.6	Bonnes Pratiques	23
5.7	Conclusion	23
6	L'instruction With ... End With	24
6.1	Introduction	24
6.2	Syntaxe	24
6.3	Exemple de Base	25
6.4	Avantages	25
6.5	Exemple avec des Objets Multiples	26
6.6	Bonnes Pratiques	26
6.7	Exemple Complet : Mise en Forme et Valeurs	27
6.8	Conclusion	27
7	Manipulation de Données	28
7.1	Calculs et Opérations de Base	28
7.2	Formatage et Boucles	29
8	Fonctions Personnalisées et Boucles	30
8.1	Fonctions Personnalisées	30
8.2	Procédure avec Boucles	31
9	Mise en Forme	32
10	Fonctions, Méthodes et Boucles en VBA	33
10.1	Fonctions Intégrées dans VBA	33
10.1.1	Fonction InStr	33
10.1.2	Fonction Chr	34
10.1.3	Fonctions de gestion des dates	34
10.1.4	Fonction Round	35
10.1.5	La méthode Offset en VBA	35
10.1.6	Autres fonctions utiles	37
10.2	Les Boucles en VBA	37

10.2.1	Boucle While...Wend	37
10.2.2	Boucle Do Until...Loop	37
10.2.3	Boucle For Each...Next	39
11	Réinitialisation de données avec les méthodes Clear	40
11.1	Méthodes principales de Clear	40
11.2	Exemple pratique	40
11.3	Applications pratiques	42
12	Manipulation de texte avec Mid, Len, Left, et Right	43
12.1	Présentation des fonctions	43
12.2	Exemple pratique	43
12.3	Applications pratiques	44
12.4	Résumé des fonctions utilisées	45
13	Classification des notes d'agence	46
13.1	Utilisation de Select Case	46
13.2	Utilisation de IIf dans une fonction personnalisée	48
13.3	Applications pratiques	48
13.4	Résumé des techniques utilisées	48
14	Les Fonctions Importées d'Excel	50
14.1	Théorie	50
14.2	Exemples	50
14.2.1	Exemple 1 : VLookup	50
14.2.2	Exemple 2 : CountIf	51
14.2.3	Exemple 3 : Sum	51
14.2.4	Exemple 4 : Average	51
14.2.5	Exemple 5 : Max et Min	52

Chapitre 1

Les Bases des Macros et des Fonctions

1.1 Introduction à VBA

VBA (Visual Basic for Applications) est un langage de programmation intégré dans les applications Microsoft (comme Excel, Word, etc.), qui permet d'automatiser des tâches et de créer des macros ou des fonctions personnalisées. Il est particulièrement utilisé dans Excel pour automatiser des calculs, manipuler des données ou créer des interfaces personnalisées.

1.2 Structure d'un Code VBA

Un code VBA est généralement constitué de deux éléments principaux : les **Subroutines** (ou macros) et les **Fonctions**.

1.2.1 Les Macros (Subroutines)

Une **macro** est une séquence d'instructions qui est exécutée lorsque vous l'appellez. Les macros sont généralement utilisées pour automatiser des tâches répétitives. Elles n'ont pas de valeur de retour, ce qui signifie qu'elles effectuent des actions sans renvoyer de résultat.

Exemple de macro :

```
1      Sub macro1()  
2      MsgBox "Hello !!"  
3  
4      'instruction1  
5      'instruction2  
6      End Sub  
7
```

Listing 1.1 – Macro simple

- **Sub macro1()** : Cela définit une macro appelée macro1.
- **MsgBox "Hello!!"** : Cette instruction affiche une boîte de message avec le texte "Hello !!".
- **'instruction1** et **'instruction2** : Ce sont des commentaires. En VBA, tout ce qui suit un apostrophe (') est ignoré lors de l'exécution du code.
- **End Sub** : Cela marque la fin de la macro.

1.2.2 Les Fonctions

Une **fonction** est similaire à une macro, mais elle permet de **retourner une valeur**. Les fonctions sont souvent utilisées pour effectuer des calculs ou manipuler des données et renvoyer un résultat.

Exemple de fonction :

```
1      Function mafonction1()  
2      'instruction1  
3      'instruction2  
4      End Function  
5
```

Listing 1.2 – Définition d'une fonction

Les fonctions peuvent également contenir des calculs, comme l'exemple ci-dessous :

```
1      Function addition(a As Integer, b As Integer)
2      As Integer
3          addition = a + b
4      End Function
```

Listing 1.3 – Fonction avec retour de valeur

Cette fonction prend deux paramètres (a et b), les additionne, puis renvoie le résultat.

1.2.3 Différence entre Sub et Function

- **Sub (Subroutine)** : Une procédure qui effectue des actions mais ne renvoie pas de valeur.
- **Function** : Une procédure qui peut effectuer des actions et qui renvoie une valeur.

1.3 Comment Utiliser les Macros et les Fonctions

1.3.1 Exécution d'une macro

Pour exécuter une **macro** dans Excel, vous pouvez :

- Lier la macro à un bouton.
- L'exécuter directement depuis l'éditeur VBA.

1.3.2 Utilisation d'une fonction dans une cellule Excel

Une **fonction** peut être utilisée dans une cellule Excel, comme une fonction Excel standard. Par exemple, une fonction `addition` que vous avez définie peut être appelée dans une cellule de la manière suivante :

`=addition(5, 10)`

Cela renverra le résultat de l'addition de 5 et 10, soit 15.

1.4 Résumé

- **Sub** : Crée une macro qui exécute des actions mais ne renvoie pas de valeur.
- **Function** : Crée une fonction qui peut effectuer des actions et renvoyer une valeur.
- Les **commentaires** (lignes commençant par ' ') sont utilisés pour expliquer le code sans affecter son exécution.

1.5 Conclusion

Le VBA est un outil puissant pour automatiser les tâches dans les applications Microsoft, comme Excel. Vous pouvez utiliser des **macros** pour exécuter des séries d'actions et des **fonctions** pour effectuer des calculs ou manipuler des données tout en renvoyant des résultats. Les commentaires dans le code sont essentiels pour documenter et clarifier les actions sans interférer avec l'exécution.

Chapitre 2

Utilisation de la fonction MsgBox en VBA

2.1 Exemple simple de MsgBox

La fonction MsgBox permet d'afficher une boîte de dialogue à l'utilisateur. Voici un exemple :

```
1 Sub bonjour()  
2 MsgBox "Bonjour ! Nous sommes le : " & Date  
3 'L'instruction Date nous donne la date du jour  
4 End Sub
```

2.2 Personnalisation de MsgBox

Vous pouvez personnaliser les boutons et les icônes de la boîte de dialogue. Exemple :

```
1 Sub bonjour_personnalise()  
2 MsgBox "Bonjour ! Nous sommes le : " & Date,  
vbYesNo + vbCritical, "Titre personnalisé"  
3 End Sub
```

Dans cet exemple :

- vbYesNo ajoute les boutons "Oui" et "Non".
- vbCritical affiche une icône d'alerte.
- Le titre de la boîte est défini par le troisième argument.

2.3 Retour à la ligne dans MsgBox

Pour insérer un retour à la ligne dans une boîte de dialogue, utilisez la fonction `Chr(10)` :

```
1 Sub msgbox_retour_ligne()  
2   MsgBox "Bonjour !" & Chr(10) & "Nous sommes le : "  
   & Date, vbOKOnly, "Message structuré"  
3 End Sub
```

2.4 Confirmation avec MsgBox

Une autre utilisation fréquente de `MsgBox` est de demander une confirmation avant d'effectuer une action. Exemple :

```
1 Sub color()  
2   If MsgBox("Voulez-vous appliquer la couleur rouge à  
   la cellule F2 ?", vbYesNo, "Confirmation") = vbYes  
   Then  
3     Range("F2").Interior.Color = RGB(255, 0, 0)  
4   Else  
5     Range("F2").ClearFormats  
6   End If  
7 End Sub
```

2.5 Exemple avancé : Mise en couleur automatique

Ce code applique des couleurs et des commentaires aux cellules d'une plage en fonction de leur valeur :

```
1 Sub applicouleur()  
2 If MsgBox("Voulez-vous appliquer la couleur et les  
commentaires ?", vbYesNo, "Confirmation") = vbNo Then  
3 Sheets("Feuil2").Range("B2:C13").Interior.Pattern =  
xlNone  
4 Sheets("Feuil2").Range("C2:C13").ClearContents  
5 Exit Sub  
6 End If  
7  
8 For i = 2 To 13  
9 If Sheets("Feuil2").Range("B" & i).Value > 0 Then  
10 Sheets("Feuil2").Range("B" & i).Interior.Color =  
RGB(0, 255, 0) ' Vert  
11 Sheets("Feuil2").Range("C" & i).Value = "Positif"  
12 ElseIf Sheets("Feuil2").Range("B" & i).Value < 0  
Then  
13 Sheets("Feuil2").Range("B" & i).Interior.Color =  
RGB(255, 0, 0) ' Rouge  
14 Sheets("Feuil2").Range("C" & i).Value = "Négatif"  
15 Else  
16 Sheets("Feuil2").Range("B" & i).Interior.Color =  
RGB(0, 0, 255) ' Bleu  
17 Sheets("Feuil2").Range("C" & i).Value = "Nul"  
18 End If  
19 Next i  
20 End Sub
```

Chapitre 3

Gestion des erreurs et affichage des informations d'un pays

Ce chapitre explore des concepts avancés en VBA liés à la gestion des erreurs, à l'interaction utilisateur via les `InputBox` et `MsgBox`, et à la manipulation de plages de données dans Excel.

3.1 Théorie : Les notions abordées

3.1.1 La gestion des erreurs

La gestion des erreurs en VBA permet de prévenir les plantages en cas d'entrée ou d'événement inattendu. L'instruction `On Error GoTo` redirige l'exécution vers un point spécifique du code lorsqu'une erreur survient.

- `On Error GoTo [nom_du_label]` : Détermine le point d'entrée en cas d'erreur.
- `Resume Next` : Ignorer l'erreur et passer à l'instruction suivante.
- `Err.Number` : Donne le numéro de l'erreur rencontrée.
- `Err.Description` : Retourne une description de l'erreur.

Exemple : Gestion d'une erreur

```
1 On Error GoTo erreur
2 ' Code risquant de générer une erreur
3
4 Exit Sub ' Sortir pour éviter d'exécuter le label
   en l'absence d'erreur
5
6 erreur:
7 MsgBox "Une erreur est survenue : " &
  Err.Description, vbCritical
```

3.1.2 Les interactions utilisateur

Les interactions utilisateur en VBA se font souvent à l'aide des fonctions suivantes :

- **MsgBox** : Affiche une boîte de message. Elle peut afficher des informations, poser des questions ou alerter l'utilisateur.
- **InputBox** : Permet de demander à l'utilisateur une entrée, qui sera ensuite traitée dans le programme.

Paramètres principaux de MsgBox :

- **Prompt** : Le texte affiché dans la boîte.
- **Buttons** : Définit les boutons et icônes (ex. vbYesNo, vbCritical).
- **Title** : Spécifie le titre de la boîte.

Exemple : Une boîte de message simple

```
1 MsgBox "Ceci est un message d'information.",
  vbInformation, "Information"
```

Paramètres principaux de InputBox :

- **Prompt** : Texte expliquant ce qui est attendu de l'utilisateur.
- **Title** : Titre de la boîte.
- **Default** : Valeur par défaut de l'entrée.

Exemple : Demander une valeur numérique

```
1 Dim valeur As Integer
2 valeur = InputBox("Veuillez saisir un entier :",
  "Entrée de données", 0)
```

3.1.3 La gestion des plages de données

En VBA, les plages de données sont manipulées à l'aide de la méthode Range. Voici quelques concepts clés :

- `Range("A1")` : Référence à une cellule spécifique.
- `Range("A1:B10")` : Référence à une plage.
- `Interior.Color` : Change la couleur de fond d'une cellule.
- `Value` : Récupère ou affecte une valeur à une cellule.

Exemple : Appliquer une couleur à une cellule

```
1 Range("A1").Interior.Color = RGB(255, 0, 0) ' Rouge
```

3.2 Exemples pratiques avec explications

3.2.1 Exemple 1 : Gestion des erreurs avec TypeVal

Le code ci-dessous montre comment demander une valeur numérique à l'utilisateur et gérer les erreurs de saisie.

```
1 Sub TypeVal()
2
3 ' En cas d'erreur, aller au message d'alerte
4 On Error GoTo msg_erreur
5
6 ' Définir la variable
7 Dim VarNum As Integer
8
9 ' Saisie de l'utilisateur
10 VarNum = InputBox("Veuillez saisir une valeur
11 numérique", _
12 "Type variable", 0)
13
14 ' Affectation de la valeur à une cellule
15 Sheets("Feuil1").Range("B3").Value = VarNum
16
17 Exit Sub
18
19 msg_erreur:
20 MsgBox "Erreur : saisie non numérique. Veuillez
21 réessayer.", _
22 vbCritical, "Alerte"
23 Call TypeVal ' Relance la procédure
24 End Sub
```

Analyse :

- Si l'utilisateur saisit une valeur invalide, le message d'alerte s'affiche et l'utilisateur doit réessayer.

— La valeur est ensuite insérée dans la cellule B3.

3.2.2 Exemple 2 : Affichage des informations d'un pays

Ce code permet d'afficher des informations spécifiques à un pays sélectionné par l'utilisateur dans une feuille Excel.

```
1 Sub FichePays()  
2  
3 Dim NumPays As Integer  
4 Dim sh As Worksheet  
5  
6 On Error GoTo msg_erreur  
7  
8 Set sh = Sheets("DATA")  
9 NumPays = InputBox("Veuillez saisir un entier entre  
10 2 et 6", _  
11 "Sélection du pays", 2)  
12  
13 If NumPays >= 2 And NumPays <= 6 Then  
14 MsgBox "Pays : " & sh.Range("A" & NumPays).Value &  
15 Chr(10) & _  
16 "Capitale : " & sh.Range("B" & NumPays).Value,  
17 vbInformation  
18  
19 Else  
20 GoTo msg_erreur  
21 End If  
22  
23 Exit Sub  
24  
25 msg_erreur:  
26 MsgBox "Erreur : valeur invalide. Veuillez  
27 réessayer.", vbCritical, "Alerte"  
28 Call FichePays  
29 End Sub
```

Analyse :

- L'utilisateur doit sélectionner un numéro correspondant à un pays dans la plage 2 à 6.
- Les informations du pays sont extraites de la feuille DATA et affichées dans une boîte de message.
- Si la saisie est incorrecte, une alerte s'affiche, et la procédure est relancée.

3.3 Conclusion

Ces exemples montrent comment gérer les interactions utilisateur et les erreurs dans un programme VBA. Les notions de `InputBox`, `MsgBox`, et de gestion des plages permettent de créer des applications interactives robustes. En combinant ces concepts, il est possible d'améliorer la fiabilité et l'expérience utilisateur des macros VBA.

Chapitre 4

Introduction aux Variables en VBA

4.1 Déclaration des Variables et Portée

En VBA, la **portée** des variables détermine où celles-ci sont accessibles dans le code :

- **Variables locales** : Déclarées dans une procédure (Sub), elles ne sont accessibles qu'à l'intérieur de cette procédure.
- **Variables globales** : Déclarées au début du module, elles sont accessibles dans toutes les procédures de ce module.

Exemple

```
1  Dim x As Integer
2
3  Sub macro1()
4      ' x est disponible ici
5      Dim y As Integer
6      ' y est disponible uniquement ici
7  End Sub
8
9  Sub macro2()
10     ' x est disponible ici
11 End Sub
```

Dans cet exemple :

- x est une variable globale, accessible dans macro1 et macro2.
- y est une variable locale, accessible uniquement dans macro1.

4.2 Variables à Longueur Fixe

VBA permet de déclarer des variables de longueur fixe pour limiter la quantité de mémoire utilisée.

```
1 Sub TestVar()  
2 Dim Tvar As String * 4 ' Longueur fixe à 4  
   caractères  
3 Dim sh As Worksheet  
4 Dim cel As Range  
5  
6 ' Affecter une valeur via une boîte de dialogue  
7 Set sh = Worksheets("Feuil1")  
8 Set cel = sh.Range("B2")  
9 Tvar = InputBox("Veuillez saisir une valeur", "Test  
   variable")  
10  
11 ' Stocker la valeur dans une cellule  
12 cel.Value = Tvar  
13 MsgBox Len(Tvar) ' Nombre de caractères  
14 End Sub
```

Cet exemple montre :

- La déclaration d'une variable Tvar avec une longueur maximale de 4 caractères.
- L'utilisation de InputBox pour récupérer une valeur saisie par l'utilisateur.
- L'affichage de la longueur de la chaîne avec la fonction Len.

4.3 Option Explicit

La directive `Option Explicit` est utilisée dans VBA pour obliger la déclaration explicite de toutes les variables avant leur utilisation dans le code. Cela signifie que chaque variable doit être déclarée à l'aide de la commande `Dim`, `Private`, ou `Public`, et une tentative d'utiliser une variable non déclarée entraînera une erreur de compilation.

Cette pratique est recommandée car elle permet de détecter les erreurs liées aux fautes de frappe dans les noms de variables, ce qui peut être difficile à repérer autrement. De plus, l'utilisation de `Option Explicit` rend le code plus lisible et prévient des comportements inattendus dus à l'utilisation de variables non définies.

4.3.1 Exemple

Voici un exemple de code avec Option Explicit activée :

```
1 Option Explicit
2
3 Sub ExempleOptionExplicit()
4 Dim x As Integer
5 x = 10
6 y = 5 ' Erreur : la variable 'y' n'est pas déclarée
7 End Sub
```

Dans cet exemple, la tentative d'utiliser la variable y sans l'avoir déclarée provoque une erreur de compilation. Si Option Explicit n'était pas activé, y aurait été automatiquement créée comme variable de type Variant, ce qui pourrait entraîner des erreurs difficiles à déboguer plus tard.

4.3.2 Avantages

- Option Explicit permet de réduire les erreurs liées aux variables mal déclarées.
- Elle rend le code plus propre et plus facile à maintenir.
- Elle aide à identifier rapidement les erreurs lors de la compilation, plutôt que lors de l'exécution du code.

Chapitre 5

Les Boucles For et While

5.1 Introduction

Les boucles permettent d'exécuter une séquence d'instructions plusieurs fois, selon une condition ou une plage de valeurs. En VBA, les boucles les plus courantes sont :

- For ... Next : utilisée pour itérer sur une plage définie de valeurs.
- Do While ... Loop : utilisée pour répéter une série d'instructions tant qu'une condition est vraie.

5.2 La Boucle For

La boucle For est utilisée lorsque le nombre d'itérations est connu à l'avance.

Structure Théorique

```
1  For [Variable] = [ValeurDépart] To [ValeurFin]
   [Step [Incrément]]
2  ' Instructions à exécuter
3  Next [Variable]
```

- **Variable** : La variable utilisée pour contrôler la boucle.
- **ValeurDépart** : La valeur initiale de la variable.
- **ValeurFin** : La valeur finale à atteindre.
- **Step** : (Optionnel) Définit l'incrément ou le décrétement. Par défaut, il vaut 1.

Exemple : Parcourir une Plage de Cellules

```
1 Sub ExempleFor()  
2 Dim i As Integer  
3 Dim sh As Worksheet  
4 Set sh = Worksheets("Feuil1")  
5  
6 For i = 1 To 10  
7   sh.Cells(i, 1).Value = "Ligne " & i  
8 Next i  
9 End Sub
```

Dans cet exemple :

- La boucle For parcourt les lignes 1 à 10 de la feuille Feuil1.
- La cellule de chaque ligne dans la colonne A reçoit la valeur "Ligne X", où X correspond au numéro de la ligne.

Exemple avec Step

```
1 Sub ExempleForStep()  
2 Dim i As Integer  
3 For i = 1 To 10 Step 2  
4   Debug.Print "Valeur : " & i  
5 Next i  
6 End Sub
```

Ici, l'instruction Step 2 incrémente la variable de 2, ce qui donne comme valeurs successives : 1, 3, 5, 7, 9.

5.3 La Boucle Do While

La boucle Do While répète une série d'instructions tant qu'une condition reste vraie.

Structure Théorique

```
1 Do While [Condition]  
2   ' Instructions à exécuter  
3 Loop
```

- **Condition** : Une expression logique ou une comparaison. La boucle s'exécute tant que cette condition est vraie.

Exemple : Addition jusqu'à une Limite

```
1 Sub ExempleWhile()  
2 Dim somme As Integer  
3 Dim i As Integer  
4 somme = 0  
5 i = 1  
6  
7 Do While somme < 20  
8   somme = somme + i  
9   i = i + 1  
10 Loop  
11  
12 MsgBox "Somme finale : " & somme  
13 End Sub
```

Dans cet exemple :

- La boucle continue tant que la somme des nombres ajoutés est inférieure à 20.
- Une boîte de message affiche la somme finale une fois la boucle terminée.

5.4 Comparaison entre For et Do While

- **Utilisez For** : lorsque le nombre d'itérations est défini ou déterminable à l'avance.
- **Utilisez Do While** : lorsque vous devez continuer à itérer jusqu'à ce qu'une certaine condition soit remplie.

5.5 Exemple Combiné : For et Do While

```
1 Sub ExempleCombine()  
2 Dim i As Integer  
3 Dim somme As Integer  
4 somme = 0  
5  
6 For i = 1 To 10  
7 If somme >= 15 Then  
8 Exit For ' Arrête la boucle si la somme atteint 15  
9 End If  
10 somme = somme + i  
11 Next i  
12  
13 MsgBox "Somme finale : " & somme  
14 End Sub
```

Cet exemple illustre :

- Une boucle For qui s'arrête prématurément à l'aide de Exit For.
- Une condition pour limiter la somme calculée.

5.6 Bonnes Pratiques

- Limitez les boucles imbriquées pour éviter des performances dégradées et des codes difficiles à lire.
- Utilisez Exit For ou Exit Do pour interrompre une boucle lorsqu'une condition est remplie.
- Assurez-vous que la condition d'arrêt dans une boucle Do While est toujours atteinte pour éviter des boucles infinies.

5.7 Conclusion

Les boucles For et While sont essentielles pour automatiser les tâches répétitives en VBA. Elles permettent de parcourir des données, effectuer des calculs ou appliquer des formats de manière efficace. Leur utilisation judicieuse est un atout pour optimiser les performances et la lisibilité du code.

Chapitre 6

L'instruction `With ... End With`

6.1 Introduction

L'instruction `With ... End With` permet de simplifier et d'optimiser le code lorsqu'on effectue plusieurs opérations sur le même objet. Elle évite de répéter l'identification de l'objet pour chaque propriété ou méthode que vous souhaitez utiliser, rendant le code plus clair et plus rapide à exécuter.

6.2 Syntaxe

La syntaxe générale est la suivante :

```
1 With [Objet]
2   .Propriété1 = Valeur1
3   .Propriété2 = Valeur2
4   .Méthode
5 End With
```

- **[Objet]** : L'objet sur lequel vous travaillez.
- **Propriété1, Propriété2, etc.** : Les propriétés que vous souhaitez modifier.
- **Méthode** : Les méthodes de l'objet que vous souhaitez appeler.

6.3 Exemple de Base

Voici un exemple simple d'utilisation pour mettre en forme une plage de cellules :

```
1 Sub MiseEnForme()  
2 With Worksheets("Feuil1").Range("A1:D5").Font  
3 .Name = "Arial"  
4 .Size = 12  
5 .Bold = True  
6 .Color = RGB(255, 0, 0) ' Rouge  
7 End With  
8 End Sub
```

Dans cet exemple, la plage de cellules A1:D5 sur la feuille Feuil1 est formatée avec :

- La police définie sur Arial.
- Une taille de 12 points.
- Un style gras.
- Une couleur rouge.

6.4 Avantages

- **Lisibilité** : Le code est plus lisible car il élimine la répétition des références à l'objet.
- **Performance** : L'objet est évalué une seule fois, ce qui peut améliorer les performances pour des objets complexes.
- **Facilité de maintenance** : Si l'objet doit être changé, il suffit de modifier une seule ligne.

6.5 Exemple avec des Objets Multiples

L'instruction `With ... End With` peut être utilisée de manière imbriquée pour travailler avec plusieurs objets :

```
1 Sub FormatMultiple()  
2 With Worksheets("Feuill1")  
3 With .Range("A1:D5")  
4 .Interior.Color = RGB(200, 200, 255) ' Couleur de  
fond  
5 .Font.Bold = True  
6 End With  
7 .Range("A1").Value = "Titre"  
8 End With  
9 End Sub
```

Ici :

- Le formatage est appliqué à la plage A1:D5.
- La cellule A1 reçoit une valeur.

6.6 Bonnes Pratiques

- Utilisez `With ... End With` uniquement si vous effectuez plusieurs opérations sur le même objet.
- Veillez à ne pas imbriquer trop de blocs `With`, car cela peut rendre le code difficile à lire.
- Combinez `With ... End With` avec des commentaires clairs pour indiquer ce que chaque section fait.

6.7 Exemple Complet : Mise en Forme et Valeurs

```
1 Sub ExempleComplet()  
2 With Worksheets("Feuil1")  
3   ' Appliquer des propriétés de mise en forme  
4   With .Range("A1:D1").Font  
5     .Name = "Verdana"  
6     .Bold = True  
7     .Size = 14  
8   End With  
9  
10  ' Insérer des valeurs  
11  .Range("A1").Value = "Produit"  
12  .Range("B1").Value = "Prix"  
13  .Range("C1").Value = "Quantité"  
14  .Range("D1").Value = "Total"  
15 End With  
16 End Sub
```

Cet exemple montre comment :

- Formater les en-têtes (A1:D1).
- Insérer des valeurs dans les cellules A1, B1, C1, et D1.

6.8 Conclusion

L'instruction `With ... End With` est une fonctionnalité essentielle pour simplifier, clarifier et optimiser le code VBA. Elle est particulièrement utile dans les projets où de nombreux objets doivent être manipulés de manière répétitive.

Chapitre 7

Manipulation de Données

7.1 Calculs et Opérations de Base

```
1 Sub test()  
2 Dim alpha As Integer, beta As Integer  
3 alpha = 6  
4 beta = 3  
5  
6 Cells(1, 1).Value = alpha + beta  
7 MsgBox alpha + beta  
8 End Sub
```

Cet exemple montre comment effectuer des calculs simples et afficher les résultats dans une cellule et une boîte de message.

7.2 Formatage et Boucles

```
1 Sub FormatNBoucle()  
2 Dim sh As Worksheet, i As Integer  
3 Set sh = Worksheets("Feuil1")  
4  
5 For i = 2 To 5  
6 If Trim(LCase(sh.Range("D" & i).Value)) = "eur" Then  
7 sh.Range("E" & i).NumberFormat = "###,###,##0.00 "  
8 ElseIf Trim(UCase(sh.Range("D" & i).Value)) = "GBP"  
Then  
9 sh.Range("E" & i).NumberFormat = "č ###,###,##0.00 "  
10 ElseIf Trim(UCase(sh.Range("D" & i).Value)) = "YEN"  
Then  
11 sh.Range("E" & i).NumberFormat = "ě ###,###,##0.00 "  
12 Else  
13 sh.Range("E" & i).NumberFormat = "###,###,##0.00 "  
14 End If  
15 Next i  
16 End Sub
```

Cet exemple montre comment :

- Utiliser une boucle For pour parcourir plusieurs cellules.
- Formater des valeurs numériques selon une devise (EUR, GBP, YEN).
- Manipuler des chaînes avec les fonctions Trim, LCase et UCase.

Chapitre 8

Fonctions Personnalisées et Boucles

8.1 Fonctions Personnalisées

```
1  Function indice(cours As Integer)
2      indice = IIf(cours > 1000, "Croissant",
3      "Décroissant")
4      End Function
5
6  Function CmNote(note As Integer)
7      CmNote = IIf(note < 8, "Faible", IIf(note < 10,
8      "Moyen", IIf(note < 15, "Bien", "Excellent")))
9      End Function
```

Les fonctions personnalisées permettent de simplifier des calculs ou des classifications répétées. Dans cet exemple :

- indice renvoie une chaîne selon la valeur d'une variable.
- CmNote évalue une note et renvoie un texte correspondant.

8.2 Procédure avec Boucles

```
1 Sub CmtBoucle()  
2 Dim sh As Worksheet, i As Integer, DerL As Integer  
3 Set sh = Worksheets("Feuil2")  
4 DerL = sh.Cells(1, 1).End(xlDown).Row  
5  
6 For i = 2 To DerL  
7 If sh.Cells(i, 2).Value < 8 Then  
8 sh.Cells(i, 7).Value = "Faible"  
9 ElseIf sh.Cells(i, 2).Value < 10 Then  
10 sh.Cells(i, 7).Value = "Moyen"  
11 ElseIf sh.Cells(i, 2).Value < 15 Then  
12 sh.Cells(i, 7).Value = "Bien"  
13 Else  
14 sh.Cells(i, 7).Value = "Excellent"  
15 End If  
16 Next i  
17 End Sub
```

Cette procédure évalue des notes et attribue un commentaire (faible, moyen, bien, excellent) dans une colonne correspondante.

Chapitre 9

Mise en Forme

```
1 Sub mise_en_forme()  
2 With Worksheets(1).Range("ma_plage1").Font  
3 .Name = "Arial"  
4 .Bold = True  
5 .Italic = True  
6 End With  
7 End Sub
```

Ce code montre comment appliquer des styles (gras, italique, police) à une plage de cellules avec la structure With...End With.

Chapitre 10

Fonctions, Méthodes et Boucles en VBA

Dans ce chapitre, nous allons explorer différentes fonctions intégrées dans VBA ainsi que les structures de boucles. Ces notions permettent de rendre les macros plus interactives et puissantes.

10.1 Fonctions Intégrées dans VBA

10.1.1 Fonction InStr

La fonction InStr retourne la position d'une chaîne de caractères dans une autre chaîne. Elle est très utile pour rechercher un caractère ou une sous-chaîne dans une chaîne donnée.

```
1 Sub InsTrt ()
2
3 MsgBox InStr(1, "test_insTr", "T", 1)
4
5 ' Le premier argument est la position de départ.
6 ' "T" est le caractère recherché.
7 ' 1 signifie que la recherche n'est pas sensible à
  la casse.
8
9 End Sub
```

Listing 10.1 – Exemple de fonction InStr

10.1.2 Fonction Chr

La fonction Chr retourne le caractère correspondant à un code ASCII donné. Cela permet de manipuler directement les caractères.

```
1 Sub ChrT()  
2  
3 MsgBox Chr(98) ' Affiche le caractère correspondant  
  au code ASCII 98 (b).  
4  
5 End Sub
```

Listing 10.2 – Exemple de fonction Chr

10.1.3 Fonctions de gestion des dates

VBA propose plusieurs fonctions pour manipuler les dates et heures :

- Date : retourne la date actuelle.
- Now : retourne la date et l'heure actuelles.
- Time : retourne uniquement l'heure actuelle.
- DateAdd : ajoute un intervalle de temps à une date donnée.

```
1 Sub dateT()  
2  
3 MsgBox Date ' Affiche la date actuelle.  
4 MsgBox Now ' Affiche la date et l'heure  
  actuelles.  
5 MsgBox Time ' Affiche uniquement l'heure  
  actuelle.  
6  
7 End Sub  
8  
9 Sub dateAddT()  
10  
11 MsgBox DateAdd("m", 1, Date) ' Ajoute un mois à  
  la date actuelle.  
12 MsgBox DateAdd("d", 1, Date) ' Ajoute un jour à  
  la date actuelle.  
13 MsgBox DateAdd("yyyy", 1, Date) ' Ajoute un an à la  
  date actuelle.  
14  
15 End Sub
```

Listing 10.3 – Exemples de fonctions de gestion des dates

10.1.4 Fonction Round

La fonction Round permet d'arrondir un nombre à un certain nombre de décimales.

```
1 Sub RoundT()  
2  
3 Dim x As Double, y As Double  
4  
5 x = 3.1458917417  
6 y = Round(x, 3) ' Arrondit à 3 décimales.  
7  
8 MsgBox y ' Affiche 3.146.  
9  
10 End Sub
```

Listing 10.4 – Exemple de fonction Round

10.1.5 La méthode Offset en VBA

La méthode Offset en VBA est utilisée pour accéder à des cellules en fonction de leur position relative par rapport à une cellule de départ. Elle est particulièrement utile pour parcourir des plages de données dynamiquement, sans avoir besoin de spécifier les coordonnées exactes.

Syntaxe

Range.Offset(Ligne, Colonne)

- **Ligne** : Nombre de lignes à se déplacer à partir de la cellule de départ.
 - Valeur positive : déplacement vers le bas.
 - Valeur négative : déplacement vers le haut.
- **Colonne** : Nombre de colonnes à se déplacer à partir de la cellule de départ.
 - Valeur positive : déplacement vers la droite.
 - Valeur négative : déplacement vers la gauche.

Exemples d'utilisation

Exemple 1 : Accéder à une cellule spécifique

```
1 Sub ExempleOffset()  
2  
3 Dim c As Range  
4  
5 ' Définir la cellule de départ  
6 Set c = Range("A1")  
7  
8 ' Accéder à une cellule 2 lignes en dessous et 1  
9   colonne à droite  
10  c.Offset(2, 1).Value = "Bonjour VBA!"  
11 End Sub
```

Listing 10.5 – Accès à une cellule spécifique

Dans cet exemple, la cellule de départ est A1. La méthode `Offset(2, 1)` permet de se déplacer de 2 lignes vers le bas et 1 colonne vers la droite, ce qui correspond à la cellule B3.

Exemple 2 : Parcourir une plage de cellules

```
1 Sub ParcourirAvecOffset()  
2  
3 Dim c As Range  
4  
5 ' Début à la cellule A1  
6 Set c = Range("A1")  
7  
8 ' Parcourir 10 lignes vers le bas  
9 Do Until c.Row > 10  
10  c.Value = "Ligne " & c.Row ' Insérer un texte dans  
   chaque cellule  
11  Set c = c.Offset(1, 0)      ' Passer à la cellule  
   de la ligne suivante  
12 Loop  
13  
14 End Sub
```

Listing 10.6 – Parcourir une plage avec Offset

Ici, la boucle commence à A1 et parcourt les 10 premières lignes. À chaque itération, la méthode `Offset(1, 0)` déplace la cellule courante d'une ligne vers le bas.

10.1.6 Autres fonctions utiles

- IsEmpty : vérifie si une cellule est vide.
- IsNumeric : vérifie si une cellule contient une valeur numérique.

```
1 Sub VideT()  
2  
3 MsgBox IsEmpty(Range("A2")) ' Vrai si la cellule  
  A2 est vide.  
4 MsgBox IsNumeric(Range("A2")) ' Vrai si la cellule  
  A2 contient un nombre.  
5  
6 End Sub
```

Listing 10.7 – Exemple de fonctions IsEmpty et IsNumeric

10.2 Les Boucles en VBA

Les boucles permettent d'exécuter des instructions de manière répétée jusqu'à ce qu'une condition soit remplie.

10.2.1 Boucle While...Wend

Cette boucle exécute les instructions tant qu'une condition donnée est vraie.

```
1 Sub MdP()  
2  
3 Dim psw As String  
4  
5 While psw <> "12345"  
6   psw = InputBox("Mot de passe ?", "Mot de passe")  
7 Wend  
8  
9 MsgBox "Mot de passe correct !"  
10  
11 End Sub
```

Listing 10.8 – Exemple de boucle While...Wend

10.2.2 Boucle Do Until...Loop

La boucle Do Until s'exécute jusqu'à ce qu'une condition soit vraie.

```
1 Sub ShortLong()  
2  
3 Dim c As Range  
4 Dim sh As Worksheet  
5 Set sh = Sheets(8)  
6  
7 ' Début de la boucle  
8 Set c = sh.Range("A2")  
9  
10 Do Until c.Value = ""  
11 If c.Offset(0, 1).Value > 0 Then  
12 c.Offset(0, 2).Value = "LONG"  
13 c.Offset(0, 2).Font.Color = RGB(0, 255, 0)  
14 ElseIf c.Offset(0, 1).Value < 0 Then  
15 c.Offset(0, 2).Value = "SHORT"  
16 c.Offset(0, 2).Font.Color = RGB(255, 0, 0)  
17 Else  
18 c.Offset(0, 2).Value = "null"  
19 c.Offset(0, 2).Font.Color = RGB(0, 0, 255)  
20 End If  
21 Set c = c.Offset(1, 0)  
22 Loop  
23  
24 End Sub
```

Listing 10.9 – Exemple de boucle Do Until...Loop

10.2.3 Boucle For Each...Next

Cette boucle permet de parcourir chaque élément d'une collection.

```
1 Sub LongShortF()  
2  
3 Dim cel As Range, sh As Worksheet, Derlig As Integer  
4  
5 Set sh = Sheets(8)  
6 Derlig = sh.Range("A1").End(xlDown).Row  
7  
8 For Each cel In sh.Range("A2:A" & Derlig)  
9 If cel.Offset(0, 1).Value > 0 Then  
10 cel.Offset(0, 2).Value = "LONG"  
11 cel.Offset(0, 2).Font.Color = RGB(0, 255, 0)  
12 ElseIf cel.Offset(0, 1).Value < 0 Then  
13 cel.Offset(0, 2).Value = "SHORT"  
14 cel.Offset(0, 2).Font.Color = RGB(255, 0, 0)  
15 Else  
16 cel.Offset(0, 2).Value = "null"  
17 cel.Offset(0, 2).Font.Color = RGB(0, 0, 255)  
18 End If  
19 Next  
20  
21 End Sub
```

Listing 10.10 – Exemple de boucle For Each...Next

Chapitre 11

Réinitialisation de données avec les méthodes `Clear`

En VBA, les méthodes `Clear`, `ClearContents` et `ClearFormats` sont utilisées pour réinitialiser ou effacer des données dans une plage de cellules. Ces méthodes permettent de supprimer soit les données, soit les formats, ou les deux.

11.1 Méthodes principales de `Clear`

- `ClearContents` : Efface uniquement le contenu des cellules, mais conserve les formats (couleurs, bordures, etc.).
- `ClearFormats` : Efface uniquement les formats des cellules, mais conserve leur contenu.
- `Clear` : Efface à la fois le contenu et les formats des cellules.

Syntaxe

```
Range.ClearContents  
Range.ClearFormats  
Range.Clear
```

11.2 Exemple pratique

Voici un exemple d'utilisation combinée de ces méthodes pour réinitialiser une colonne de données dans une feuille de calcul.

```

1      Sub ReInit()
2
3      Dim DerL As Integer
4      Dim sh As Worksheet
5
6      ' Affecter des valeurs aux variables
7      Set sh = Worksheets(7)
8      DerL = sh.Cells(1, 1).End(xlDown).Row
9
10     ' Réinitialiser les colonnes de C2 à la dernière
11     ligne remplie
12     sh.Range("C2:C" & DerL).ClearContents ' Efface le
13     contenu
14     sh.Range("C2:C" & DerL).ClearFormats ' Efface les
15     formats
16     sh.Range("C2:C" & DerL).Clear        ' Efface
17     contenu + formats
18
19     End Sub

```

Listing 11.1 – Exemple de réinitialisation avec Clear

Explications détaillées

— Déclaration des variables :

- DerL : Variable qui détermine la dernière ligne non vide de la colonne A.
- sh : Objet représentant la feuille de calcul numéro 7.

— Détection de la dernière ligne remplie :

```

1      DerL = sh.Cells(1, 1).End(xlDown).Row
2

```

Cette ligne utilise End(xlDown) pour détecter la dernière cellule non vide à partir de la cellule A1.

— Effacement ciblé :

- ClearContents supprime uniquement les valeurs de la plage C2:C&DerL.
- ClearFormats supprime uniquement les formats de la plage.
- Clear combine les deux opérations.

11.3 Applications pratiques

Ces méthodes sont particulièrement utiles pour :

- Réinitialiser une plage de données avant une nouvelle saisie.
- Supprimer des formats appliqués par erreur tout en conservant les données.
- Nettoyer des feuilles de calcul en préparation d'une nouvelle analyse.

Chapitre 12

Manipulation de texte avec Mid, Len, Left, et Right

Ce chapitre explore diverses fonctions VBA utilisées pour manipuler des chaînes de caractères, en particulier lorsqu'il s'agit de nettoyer ou de transformer des données textuelles dans une feuille Excel.

12.1 Présentation des fonctions

- **Mid** : Extrait une sous-chaîne d'une chaîne à partir d'une position donnée.
 - `Mid(chaine, départ, longueur)`
- **Len** : Renvoie la longueur totale d'une chaîne.
- **Left** : Extrait les premiers caractères d'une chaîne.
 - `Left(chaine, longueur)`
- **Right** : Extrait les derniers caractères d'une chaîne.
 - `Right(chaine, longueur)`

12.2 Exemple pratique

L'exemple suivant montre comment utiliser ces fonctions pour modifier des chaînes de texte en fonction de conditions spécifiques, telles que la présence d'un préfixe "X_" ou une longueur excessive.

```
1 Sub CleanText()  
2  
3 Dim cel As Range, sh As Worksheet  
4 Set sh = Worksheets(1)  
5  
6 For Each cel In sh.Range("A1:A10") ' Parcourt les  
   cellules de la colonne A  
7  
8   ' Condition 1 : Si la chaîne commence par "X_"  
9   If Left(cel.Value, 2) = "X_" Then  
10    cel.Value = Right(cel.Value, Len(cel.Value) - 2)  
11   End If  
12  
13   ' Condition 2 : Si la chaîne contient plus de 7  
   caractères  
14   If Len(cel.Value) > 7 Then  
15    cel.Value = Right(cel.Value, Len(cel.Value) - 2)  
16   End If  
17  
18 Next cel  
19  
20 End Sub
```

Listing 12.1 – Exemple avec Left, Right, Mid, et Len

Explications détaillées

- **Condition** If Left(cel.Value, 2) = "X_" :
 - Left(cel.Value, 2) extrait les deux premiers caractères de la chaîne.
 - La condition vérifie si ces caractères correspondent à "X_".
 - Right(cel.Value, Len(cel.Value) - 2) extrait les caractères restants après suppression des deux premiers.
- **Condition** If Len(cel.Value) > 7 :
 - Len(cel.Value) calcule la longueur totale de la chaîne.
 - Si cette longueur dépasse 7, les deux premiers caractères sont supprimés à l'aide de Right.

12.3 Applications pratiques

Ce type de manipulation est utile pour :

- Nettoyer des données textuelles avec des préfixes indésirables.

- Réduire la longueur des chaînes pour répondre à des contraintes spécifiques.
- Préparer des données textuelles pour des analyses ou transformations ultérieures.

12.4 Résumé des fonctions utilisées

- Left : Extrait les premiers caractères d'une chaîne.
- Right : Extrait les derniers caractères d'une chaîne.
- Mid : Extrait une sous-chaîne d'une chaîne.
- Len : Renvoie la longueur totale d'une chaîne.

Ces fonctions, combinées avec des conditions comme If, permettent une manipulation efficace et ciblée des données textuelles en VBA.

Chapitre 13

Classification des notes d'agence

Ce chapitre présente une procédure et une fonction VBA permettant de classer les notes d'une agence en différentes catégories qualitatives en fonction de leurs valeurs.

13.1 Utilisation de Select Case

La procédure suivante, SignAg, utilise une boucle et une structure conditionnelle Select Case pour attribuer une classification textuelle basée sur la note donnée.

```
1 Sub SignAg()  
2  
3 ' Déclaration de variables  
4 Dim NoteAgence As String  
5 Dim SignNote As String  
6 Dim sh As Worksheet  
7 Dim i As Integer, Derligne As Integer  
8  
9 ' Définir la feuille de calcul  
10 Set sh = Sheets(4)  
11  
12 ' Identifier la dernière ligne de données  
13 Derligne = sh.Cells(200, 1).End(xlUp).Row  
14  
15 ' Boucle à travers chaque ligne pour analyser la  
16 note  
17 For i = 2 To Derligne
```



```
18     NoteAgence = sh.Cells(i, 2).Value ' Lecture de la
19     note dans la colonne B
20
21     ' Classification à l'aide de Select Case
22     Select Case NoteAgence
23
24     Case "AAA"
25         SignNote = "Prime"
26
27     Case "AA", "AA-", "AA+"
28         SignNote = "High grade"
29
30     Case "A", "A-", "A+"
31         SignNote = "Upper medium grade"
32
33     Case "BBB", "BBB-", "BBB+"
34         SignNote = "Lower medium grade"
35
36     Case Else
37         SignNote = "Note indisponible"
38
39     End Select
40
41     ' Stockage de la classification dans la colonne D
42     sh.Cells(i, 4).Value = SignNote
43
44     Next i
45
46 End Sub
```

Listing 13.1 – Procédure SignAg

Explications

- **Select Case** : Permet de tester plusieurs cas pour une variable donnée.
- **Case Else** : Définit une action par défaut si aucun des cas spécifiés n'est rempli.
- Les résultats sont enregistrés dans la colonne D de la feuille.

13.2 Utilisation de IIf dans une fonction personnalisée

La fonction SignNoteF est une alternative compacte à la procédure SignAg. Elle utilise des expressions imbriquées IIf pour déterminer la classification.

```
1  Function SignNoteF(NoteAg As String)
2
3      SignNoteF = IIf(NoteAg = "AAA", "Prime", _
4      IIf(NoteAg = "AA" Or NoteAg = "AA+" Or NoteAg =
5      "AA-", "High grade", _
6      IIf(NoteAg = "A" Or NoteAg = "A+" Or NoteAg = "A-",
7      "Upper grade", _
8      IIf(NoteAg = "BBB" Or NoteAg = "BBB+" Or NoteAg =
9      "BBB-", "Lower Medium grade", _
      "NON DISPONIBLE"))))
10 End Function
```

Listing 13.2 – Fonction SignNoteF

Explications

- IIf : Une fonction VBA qui retourne une valeur en fonction d'une condition.
- Les expressions IIf imbriquées remplacent la structure Select Case.
- La fonction retourne la classification correspondant à la note passée en paramètre.

13.3 Applications pratiques

Ces outils sont particulièrement utiles pour :

- Classifier les notes d'une agence de manière automatique.
- Gérer des jeux de données volumineux.
- Fournir une vue synthétique des données dans une feuille Excel.

13.4 Résumé des techniques utilisées

- Select Case : Simplifie la gestion des conditions multiples.
- IIf : Permet d'imbriquer des conditions dans une fonction unique.

- Boucles For . . . Next : Automatisent l'application des règles sur plusieurs lignes.

Ces deux approches offrent des solutions flexibles pour le traitement de données textuelles dans Excel à l'aide de VBA.

Chapitre 14

Les Fonctions Importées d'Excel

14.1 Théorie

En VBA, il est possible d'utiliser une large gamme de fonctions natives d'Excel pour effectuer des calculs directement dans le code. Ces fonctions sont accessibles via l'objet `WorksheetFunction`, qui permet d'accéder aux fonctions Excel comme `VLookup`, `Sum`, `CountIf`, et bien d'autres.

Ces fonctions sont très utiles pour effectuer des calculs complexes ou des recherches de données directement dans le code VBA, sans avoir besoin de manipuler manuellement les cellules. Elles sont appelées de la manière suivante :

- `WorksheetFunction.FonctionExcel(argument1, argument2, ...)`

Les fonctions les plus courantes comprennent :

- `VLookup` : Recherche une valeur dans une table et retourne une valeur correspondante.
- `Sum` : Additionne une plage de valeurs.
- `CountIf` : Compte les cellules qui satisfont à un critère spécifique.
- `Average` : Calcule la moyenne des valeurs dans une plage.
- `Max` et `Min` : Renvoient respectivement la valeur maximale et minimale d'une plage.

14.2 Exemples

14.2.1 Exemple 1 : `VLookup`

La fonction `VLookup` recherche une valeur dans la première colonne d'une plage et renvoie une valeur correspondante à partir d'une autre colonne.

```
1 data.Cells(i, 5).Value =  
WorksheetFunction.VLookup(Mid(data.Cells(i,  
1).Value, 5, 3), ref.Columns("A:C"), 3, False)
```

Cette ligne recherche une sous-chaîne dans la colonne A de la feuille ref, et retourne la valeur correspondante dans la troisième colonne de la plage A:C.

14.2.2 Exemple 2 : CountIf

La fonction CountIf compte le nombre de cellules dans une plage qui répondent à un critère spécifique.

```
1 data.Cells(i, 8).Value =  
WorksheetFunction.CountIf(Columns(4), data.Cells(i,  
4).Value)
```

Ici, CountIf compte combien de fois la valeur de la cellule D(i) apparaît dans la colonne D.

14.2.3 Exemple 3 : Sum

La fonction Sum additionne les valeurs dans une plage de cellules.

```
1 .Cells(Derligne + 2, 7).Value =  
WorksheetFunction.Sum(.Range("G2:G" & Derligne))
```

Cette ligne additionne toutes les valeurs de la colonne G de la ligne 2 à la dernière ligne (Derligne), et place le résultat dans la cellule G(Derligne + 2).

14.2.4 Exemple 4 : Average

La fonction Average calcule la moyenne des valeurs dans une plage de cellules.

```
1 data.Cells(i, 9).Value =  
WorksheetFunction.Average(data.Range("C2:C" &  
Derligne))
```

Cette ligne calcule la moyenne des valeurs dans la plage C2:C(Derligne) et place le résultat dans la cellule I(i).

14.2.5 Exemple 5 : Max et Min

Les fonctions Max et Min renvoient respectivement la valeur maximale et minimale d'une plage.

```
1 data.Cells(i, 10).Value =  
WorksheetFunction.Max(data.Range("C2:C" & Derligne))  
2 data.Cells(i, 11).Value =  
WorksheetFunction.Min(data.Range("C2:C" & Derligne))
```

Les deux lignes ci-dessus retournent respectivement la valeur maximale et minimale de la plage C2:C(Derligne).