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'appelez. 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)
   As Integer
2        addition = a + b
3        End Function
4
```

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 :

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

2.2 Personnalisation de MsgBox

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

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

```
On Error GoTo erreur
' Code risquant de générer une erreur

Exit Sub ' Sortir pour éviter d'exécuter le label en l'absence d'erreur

erreur:
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

```
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

```
Dim valeur As Integer
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

```
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.

```
Sub TypeVal()

' En cas d'erreur, aller au message d'alerte
On Error GoTo msg_erreur

' Définir la variable
Dim VarNum As Integer

' Saisie de l'utilisateur
VarNum = InputBox("Veuillez saisir une valeur numérique", _
"Type variable", 0)

' Affectation de la valeur à une cellule
Sheets("Feuil1").Range("B3").Value = VarNum

Exit Sub

msg_erreur:
MsgBox "Erreur : saisie non numérique. Veuillez réessayer.", _
vbCritical, "Alerte"
Call TypeVal ' Relance la procédure
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
    2 et 6", _
10  "Sélection du pays", 2)
11
12  If NumPays >= 2 And NumPays <= 6 Then
13  MsgBox "Pays : " & sh.Range("A" & NumPays).Value &
    Chr(10) & _
14  "Capitale : " & sh.Range("B" & NumPays).Value,
    vbInformation
15  Else
16  GoTo msg_erreur
17  End If
18
19  Exit Sub
20
21  msg_erreur:
22  MsgBox "Erreur : valeur invalide. Veuillez
    réessayer.", vbCritical, "Alerte"
23  Call FichePays
24  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

```
Dim x As Integer

Sub macro1()
' x est disponible ici
Dim y As Integer
' y est disponible uniquement ici
End Sub

Sub macro2()
' x est disponible ici
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`.

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 décrément. Par défaut, il vaut 1.

Exemple : Parcourir une Plage de Cellules

```
Sub ExempleFor()
Dim i As Integer
Dim sh As Worksheet
Set sh = Worksheets("Feuil1")

For i = 1 To 10
sh.Cells(i, 1).Value = "Ligne " & i
Next i
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`

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

```
Do While [Condition]
' Instructions à exécuter
Loop
```

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

Exemple : Addition jusqu'à une Limite

```
Sub ExempleWhile()
Dim somme As Integer
Dim i As Integer
somme = 0
i = 1

Do While somme < 20
somme = somme + i
i = i + 1
Loop

MsgBox "Somme finale : " & somme
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`

```
Sub ExempleCombine()
Dim i As Integer
Dim somme As Integer
somme = 0

For i = 1 To 10
If somme >= 15 Then
Exit For ' Arrête la boucle si la somme atteint 15
End If
somme = somme + i
Next i

MsgBox "Somme finale : " & somme
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 :

```
With [Objet]
.Propriété1 = Valeur1
.Propriété2 = Valeur2
.Méthode
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 :

```
Sub MiseEnForme()
With Worksheets("Feuil1").Range("A1:D5").Font
.Name = "Arial"
.Size = 12
.Bold = True
.Color = RGB(255, 0, 0) ' Rouge
End With
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("Feuil1")
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

```
Sub ExempleComplet()
With Worksheets("Feuil1")
' Appliquer des propriétés de mise en forme
With .Range("A1:D1").Font
.Name = "Verdana"
.Bold = True
.Size = 14
End With

' Insérer des valeurs
.Range("A1").Value = "Produit"
.Range("B1").Value = "Prix"
.Range("C1").Value = "Quantité"
.Range("D1").Value = "Total"
End With
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

```
    Sub test()
    Dim alpha As Integer, beta As Integer
    alpha = 6
    beta = 3

    Cells(1, 1).Value = alpha + beta
    MsgBox alpha + beta
    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",
    "Décroissant")
3     End Function
4
5     Function CmNote(note As Integer)
6     CmNote = IIf(note < 8, "Faible", IIf(note < 10,
    "Moyen", IIf(note < 15, "Bien", "Excellent")))
7     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`.