Débutez en VBA Word

Par Olivier Lebeau

Date de publication : 25 janvier 2009

Cet article traite du VBA spécifique de Word. Il est destiné à un public ayant quelques bases en VBA (bases disponibles dans les tutos précédents). Vous trouverez dans cet articles exercices et corrigés.

I - Introduction

Alors que les cinq articles précédents traitaient des généralités VBA, cet article est le premier qui ne va traiter que du VBA pour Word. Dans cet article, nous allons utiliser l'application Word et apprendre à travailler avec les documents.

Si vous n'avez aucune bases en VBA et que vous souhaitez vous initier à ce langage, je vous conseille l'article suivant : **Débuter en VBA**

Nous allons dans les prochains tutoriels utiliser nombre d'objets. Un objet est une entité qui possède des propriétés, contient des méthodes et peut réagir aux évènements.

Nous pourrions comparer un objet à une voiture.
Une voiture possède des propriétés, sa couleur, sa marque, son modèle, sa cylindrée. Elle possède des méthodes, elle peut rouler, freiner, tourner, s'arrêter. Finalement, elle peut réagir à certains évènements comme avertir le conducteur de la présence d'une panne.

I-A - Passage des arguments

Lorsque nous utilisons des méthodes, nous sommes parfois amenés à leur passer des arguments, ce qui se fera sous la forme suivante :

```
Objet.Méthode Argument1:="Valeur", Argument4:="Valeur"
```

Le nom de l'argument est suivi de la combinaison "**:=**" et ensuite la valeur. Nous pourrions le faire sans nommer les arguments :

```
Objet.Méthode "Valeur",,,"Valeur"
```

Mais il est plus commode de les nommer, vous ne serez pas amenés à faire des erreurs liées à l'ordre et au nombre des arguments. Dans cet exemple, il n'y a que quatre arguments, imaginez avec 10 arguments ou plus.

II - L'objet Application Word

Certaines mauvaises langues diront que VBA n'est pas un langage de programmation orienté objet.
Si vous utilisez le VBA, vous travaillez avec des objets. Le modèle Word possède des collections, des méthodes, des propriétés et réagit aux évènements.

L'application Word est également un Objet que vous allez utiliser.
Comme nous le savons déjà, le VBA a besoin d'une application hôte pour pouvoir être utilisé.
Si vous utilisez Word comme application hôte, vous ne devez pas déclarer l'objet Application, vous ne devrez le faire que si vous utilisez une autre application hôte que Word.

Nous avons vu que Word possède des collections, la plus importante d'entre elles est la collection **Documents**. Nous étudierons cette collection au chapitre suivant.

Comment déclarer une variable objet en VBA ?
Pour déclarer un objet, la méthode est la même que pour une variable de type simple, on utilise Dim, Private ou Public.

```
Dim objWApp As Word.Application
```

Pour affecter un objet à sa variable, nous devons utiliser l'instruction Set (contrairement aux variables simples) si l'objet n'est pas déjà crée, il faut également utiliser de New, CreateObject ou GetObject.

Création d'une nouvelle instance en Early Binding

```
Dim objWApp As Word.Application
'Crée un nouvelle instance de l'application
Set objWApp = New Word.Application
```

On peut le faire immédiatement lors de la déclaration

Création d'une nouvelle instance en Early Binding (2)

```
'Crée un nouvelle instance de l'application lors de la déclaration
Dim objWApp As New Word.Application
```

Utilisation d'une instance existante en Early Binding

```
Dim objWApp As Word.Application
'Utilise instance existante de l'application
Set objWApp = Word.Application
```

Création d'une nouvelle instance en Late Binding

```
Dim objWApp As Object
'Crée une nouvelle instance de l'application
Set objWApp = CreateObject("Word.Application")
```

Utilisation d'une instance existante en Late Binding

```
Dim objWApp As Object
'Crée une nouvelle instance de l'application
Set objWApp = GetObject(,"Word.Application")
```

*La différence entre **Early Binding** et **Late Binding** est le moment du référencement de la bibliothèque. Dans le Early Binding, la bibliothèque est ajoutée au projet par le menu **Outils** => **Références**. Vous bénéficiez alors de l'**IntelliSense** qui est l'auto complétion de votre code.*
*Dans le **Late Binding**, on ne fait référence à la bibliothèque que dans le code et pas avant, dans ce cas, vous ne bénéficiez pas de l'**IntelliSense** et les erreurs dans votre code peuvent être plus nombreuses.*

Lorsque vous avez fini d'utiliser une variable objet, vous devez libérer les objets avec la commande :

```
Set objWApp = Nothing
```

Si vous le ne faites pas, vous gardez inutilement en mémoire l'espace nécessaire pour une instance.

II-A - Propriétés

Dans les propriétés de Word, certaines sont en lecture seule et permettent de récupérer certaines informations alors que d'autres sont en lecture/écriture. Dans ce dernier cas, vous pouvez intervenir pour les modifier.

II-A-1 - ActiveDocument

Cette propriété en lecture seule renvoie le document actif.

ActiveDocument

```
Sub CheminDocumentActif()
Debug.Print Application.ActiveDocument.FullName
End Sub
```

Comme mentionné plus tôt, nous aurions pu écrire :

ActiveDocument

```
Sub CheminDocumentActif()
Debug.Print ActiveDocument.FullName
End Sub
```

II-A-2 - ActivePrinter

Cette propriété renvoie le nom de l'imprimante active du système, cette propriété est en lecture/écriture.

ActivePrinter

```
Sub NomImprimante()
Debug.Print Application.ActivePrinter
End Sub
```

Ce code n'a aucun intérêt, par contre, mémoriser le nom de l'imprimante active, attribuer une nouvelle imprimante pour un travail particulier et restaurer l'imprimante par défaut ensuite peut s'avérer très intéressant.

Changer l'imprimante active

```
Sub ChangerImprimante()
'Déclaration des variables
Dim strOldPrinter As String
'Récupération du nom de l'imprimante active
strOldPrinter = Application.ActivePrinter
'Changement de l'imprimante
Application.ActivePrinter = "PDF Creator"
'Impression
ActiveDocument.PrintOut
'Attendre la fin de l'impression
DoEvents
'Restaurer l'imprimante de départ
ActivePrinter = strOldPrinter

End Sub
```

II-A-3 - Caption

Cette propriété vous permet de modifier le titre contenu dans la barre de titre de Word. Si vous voulez donner une touche personnelle à Word, cette propriété vous permettra d'y parvenir.

Changer titre de l'application

```
Sub ChangerTitreBarre()
Application.Caption = "www.developpez.com"
End Sub
```

II-A-4 - DisplayRecentFiles

Propriété en lecture/écriture autorisant ou non l'affichage des fichiers récents.

Afficher les fichiers récents

```
Sub AfficherFichiersRecents()
Application.DisplayRecentFiles = False
End Sub
```


Supprimer l'affichage efface le contenu de la liste, même si on le rétablit ensuite.

II-A-5 - FileDialog

Renvoie un objet FileDialog qui représente une instance unique d'une boîte de dialogue Fichier. Cette propriété est très utile lorsque vous devez choisir un fichier ou un répertoire.

```
Sub AfficherFileDialog()
Dim dlg As FileDialog
Dim strPath As String

Set dlg = Application.FileDialog(msoFileDialogFolderPicker)
With dlg
    .InitialFileName = "C:\temp\"
    .AllowMultiSelect = False
    .Title = "www.developpez.com"
    .Show
End With

strPath = dlg.SelectedItems(1)

Set dlg = Nothing

End Sub
```

Il existe quatre types de FileDialog :

msoFileDialogFilePicker	Sélection de fichier
msoFileDialogFolderPicker	Sélection de répertoire
msoFileDialogOpen	Boîte de dialogue **Ouvrir**
msoFileDialogSaveAs	Boîte de dialogue **Enregistrer Sous**

Lorsque vous utilisez l'objet FileDialog, vous pouvez spécifier si l'utilisateur pourra sélectionner plusieurs fichiers ou un seul à la fois; vous pouvez également spécifier un répertoire de départ.

Dans l'exemple que je viens de donner, vous aurez probablement remarqué la présence d'un With.....End With. Cette structure permet pour un même objet d'exécuter une série d'instructions :

Avec with

```
With dlg
    .InitialFileName = "C:\temp\"
    .AllowMultiSelect = False
    .Title = "www.developpez.com"
    .Show
End With
```

est équivalent à :

Sans with

```
dlg.InitialFileName = "C:\temp\"
dlg.AllowMultiSelect = False
```

Sans with
```
dlg.Title = "www.developpez.com"
dlg.Show
```

Exercice :
Créez une fonction qui va ouvrir un objet FileDialog, passez en argument le titre de votre boîte de dialogue ainsi que le répertoire de départ. Cette fonction renverra le nom du fichier choisi.

Nous n'avons utilisé qu'un seul élément sélectionné dans notre liste, si vous le spécifiez, vous pouvez autoriser une sélection multiple. Ci-dessous, vous trouverez un exemple de code pour une sélection multiple et la façon de récupérer les différents éléments choisis.

Avec une sélection multiple
```
Sub ListeSelectionMultiple()
Dim oDlg As FileDialog
Dim varSelected As Variant

Set oDlg = Application.FileDialog(msoFileDialogFilePicker)
With oDlg
    .AllowMultiSelect = True
    .Show
End With

For Each varSelected In oDlg.SelectedItems
    Debug.Print varSelected
Next varSelected

Set oDlg = Nothing

End Sub
```

La première partie reste la même, seule la récupération diffère. La variable qui va être utilisée pour la récupération des éléments doit être du type Variant. Ensuite, on récupère chaque élément à l'aide d'une boucle.
La valeur récupérée renvoie le chemin complet des fichiers sélectionnés.

II-A-6 - NormalTemplate

Cette propriété renvoie le modèle Normal, communément appelé "Normal.dot"

NormalTemplate
```
Sub CheminDuNormalPointDot()
Debug.Print Application.NormalTemplate.FullName
End Sub
```

Ce code vous donne le chemin complet du **normal.dotm**.

II-A-7 - Options

Cette propriété donne accès aux différentes options de l'environnement de Word. Vous trouverez ci-dessous quelques options disponibles, la liste complète est très longue.

Options Word
```
Sub OptionsDeWord()
With Application.Options
    .AllowDragAndDrop = False
    .AllowReadingMode = True
    .AutoFormatAsYouTypeReplaceHyperlinks = True
```

Options Word
```
    .CheckGrammarAsYouType = True
    .IgnoreUppercase = True
End With
End Sub
```

II-A-8 - ScreenUpdating

Cette propriété permet de geler l'affichage lorsqu'on lui affecte la valeur **False**.

*Cette propriété est particulièrement intéressante si vous travaillez avec l'objet **Selection**.*

```
Application.ScreenUpdating = False
...
...
Application.ScreenUpdating = True
```

Si vous voulez obtenir un rafraichissement occasionnel de l'affichage utilisez :

```
Application.ScreenRefresh
```

II-A-9 - System

Cette propriété permet de récupérer quelques informations sur le système, le type d'OS, l'espace disque disponible ...

```
Sub ProprietesSystem()
With Application.System
    Debug.Print .OperatingSystem 'Version OS installée
    Debug.Print .HorizontalResolution 'Résolution verticale
    Debug.Print .VerticalResolution 'Résolution horizontale
End With
End Sub
```

Vous remarquerez que l'on peut dans une structure With ... End With ajouter des fonctions comme un Debug.Print dans l'exemple ou une affectation à une variable.

II-A-10 - Visible

Cette propriété permet détermine si l'objet application est visible ou pas. Elle principalement utilisée lorsque l'on utilise Word à partir d'une autre application.

II-B - Méthodes

Après les propriétés, nous allons voir quelques méthodes de l'objet Application.

II-B-1 - ChangeFileOpenDirectory

Par défaut, si vous ne spécifiez pas de répertoire lorsque vous tentez d'ouvrir un fichier, c'est l'endroit où Word va chercher. Il est possible de modifier ce répertoire.

```
Sub ModifierCheminParDefaut()
```

```
Application.ChangeFileOpenDirectory "C:\temp\"
End Sub
```

Exercice :
Avec ce que nous avons vu précédemment, changez le répertoire par défaut et ensuite, restaurez-le.

*Ne pas confondre **ChangeFileOpenDirectory** et **DefaultFilePath** l'un est volatile alors que le second est permanent. Le changement produit par le premier s'estompe lorsque Word est fermé.*

II-B-2 - Conversion d'unités

Méthode	Résultat
CentimetersToPoints()	Conversion de centimètres en points, prend comme argument un Double
InchesToPoints()	Conversion de pouces en points, prend comme argument un Double
LinesToPoints()	Converti une ligne en points, une ligne = 12 points, prend comme argument un Single
MilimetersToPoints()	Conversion de millimètres en points, 1 mm = 2,835 points, prend comme argument un Single
PicasToPoint()	Conversion de picas en points, 1 pica = 12 points, prend comme argument un Single
PixelsToPoint()	Conversion de pixels en points, prend deux arguments, le premier : le nombre de pixels Single, le second, un Boolean facultatif. Le second correspond à Vertical Oui ou Non
PointsToCentimeters()	Conversion de points en centimètre, 1 cm = 28,35 points, prend comme argument un Single.
PointsToInches()	Conversion de points en pouces, 1 pouce = 72 points, prend comme argument un Single.
PointsToLines()	Conversion de points en lignes, 1 ligne = 12 points, prend comme argument un Single.
PointsToMilimeters()	Conversion de points en millimètres, prend comme argument un Single.
PointsToPicas()	Conversion de points en picas, prend en argument un Single.
PointsToPixels()	Conversion de points en pixels, prend deux arguments, un Single et un Boolean si les pixels sont verticaux.

II-B-3 - OnTime

Si vous craignez de rester trop longtemps au boulot, cette propriété est pour vous.
Pour son utilisation, vous avez deux possibilités, soit vous utilisez une heure précise soit vous spécifiez un délai.

La méthode OnTime permet de lancer une routine à un moment prédéterminé.

Heure Précise

```
Application.Ontime When:="16:00:00", Name:="MaProcedure"
```

Délai

```
Application.Ontime When:=Now + TimeValue("00:10:00"), Name:="MaProcédure"
```

L'argument When est passé en format texte.

II-B-4 - Quit

C'est la méthode qui possède le nom le plus clair. Cette méthode ferme l'application; n'oubliez pas de libérer l'instance de votre objet une fois que vous avez quitté l'application.

```
Application.Quit
```

Exercice :
En vous basant sur les fonctions VBA pour la création d'un objet, déclarez un objet Word, affectez-lui une valeur et fermez le tout.

II-C - Événements

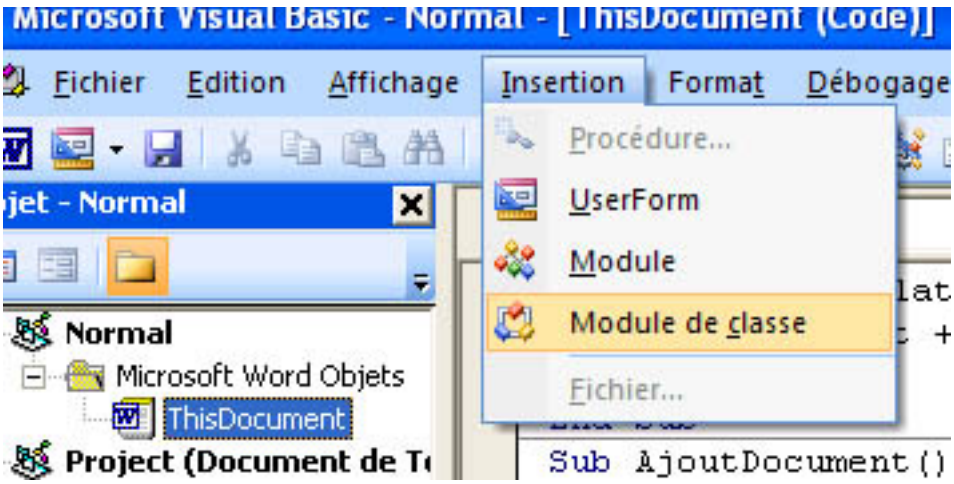
Alors que la mise en œuvre des événements pour les documents est très simple, la mise en œuvre pour l'objet Application se révèle un peu plus complexe.

II-C-1 - Gestion des événements

Deux méthodes seront abordées pour la gestion des événements. Vous pouvez utiliser un module de classe ou directement dans votre module "ThisDocument". En utilisant un module de classe, vous pouvez réutiliser votre code à souhait sans devoir l'écrire pour chacun de vos projets.

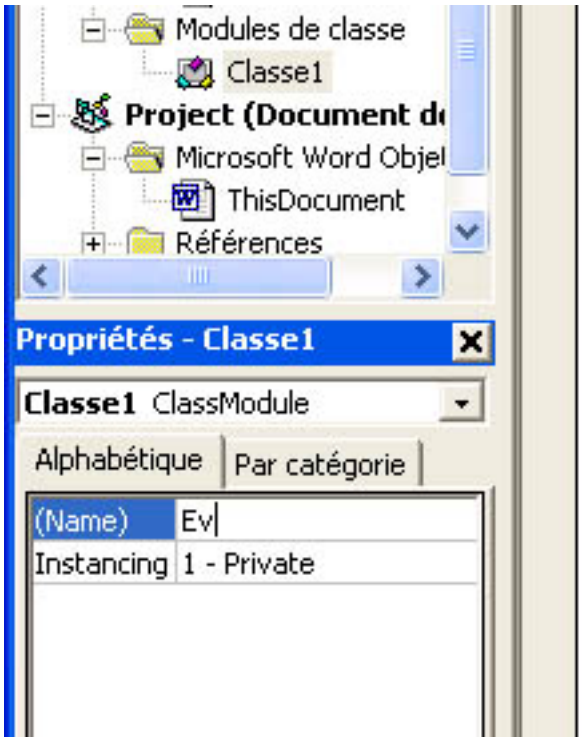
II-C-1-a - Module de classe

Pour la gestion des événements, nous devons créer un module de classe pour y déclarer un objet application pour la gestion des événements.



Ajout d'un module de classe

Vous aller renommer ce module en **EventClassModule**.



Renommer en EventClassModule

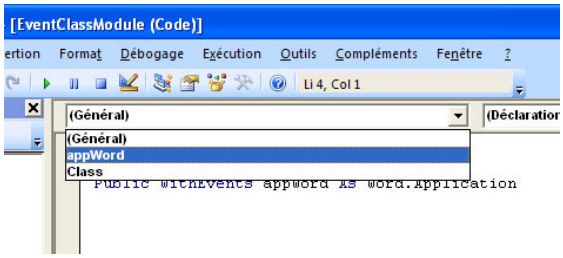
Nous allons déclarer notre application

Déclaration de notre objet application

```
Public WithEvents appWord As Word.Application
```

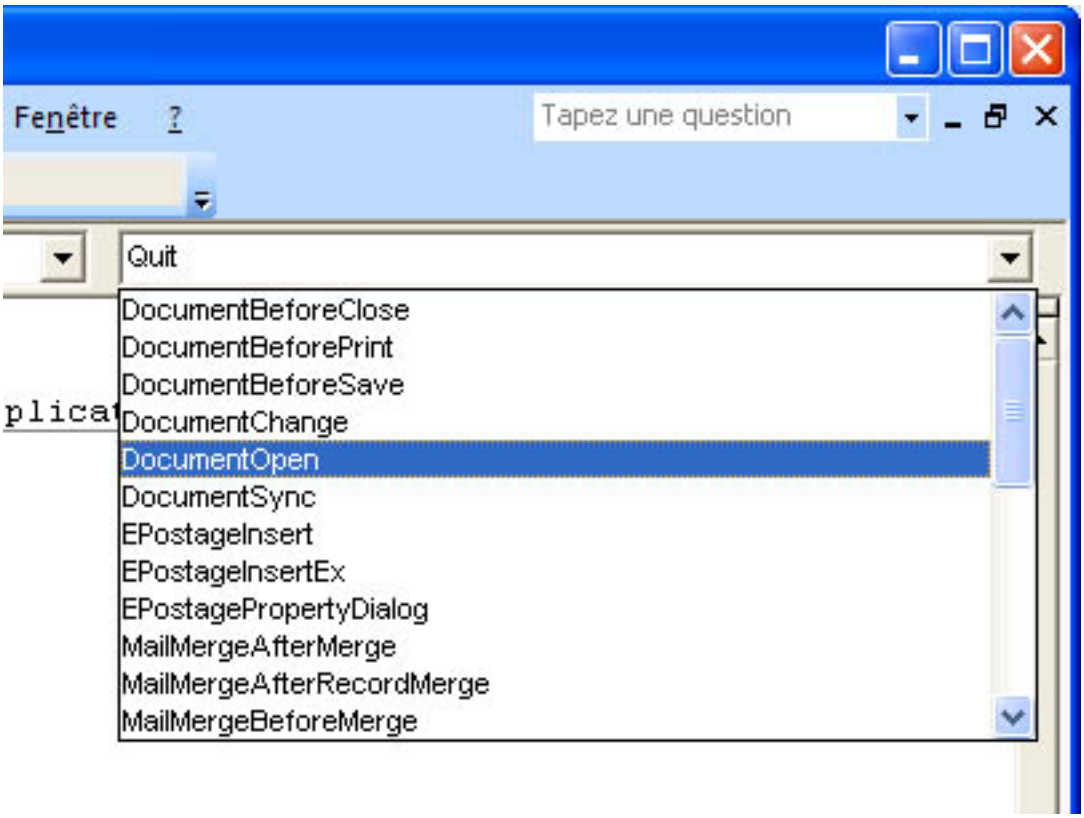
*WithEvents : mot clé indiquant que l'argument varname est une variable objet utilisée pour répondre aux événements déclenchés par un objet **ActiveX**. Valide uniquement dans des modules de classe.*

Dès que votre objet Application est déclaré, les événements sont disponibles dans la liste.



L'objet Application

Si vous choisissez l'objet Application, vous pouvez choisir un événement qui lui est associé.



Les événements

Les événements liés à l'application doivent se trouver dans le module de classe.

II-C-1-b - Initialisation du module de classe

Ce n'est pas terminé, vous devez instancier le module de classe. L'idéal est de le faire sur le modèle où vous utilisez la gestion d'événement, ou bien directement dans le **normal.dotm**, sur l'événement **Document_New()**.

Pour activer la gestion des événements.

```
Dim MyApp As New EventsClassModule
Sub Document_New()
Set MyApp.appWord = Word.Application
End Sub
```

Vous pouvez initialiser la gestion d'événement comme bon vous semble.

II-C-1-c - Directement dans votre module (sans module de classe)

Comme pour le module de classe, il faut déclarer un objet Application.

Déclaration de l'application
```
Public WithEvents App As Word.Application
```

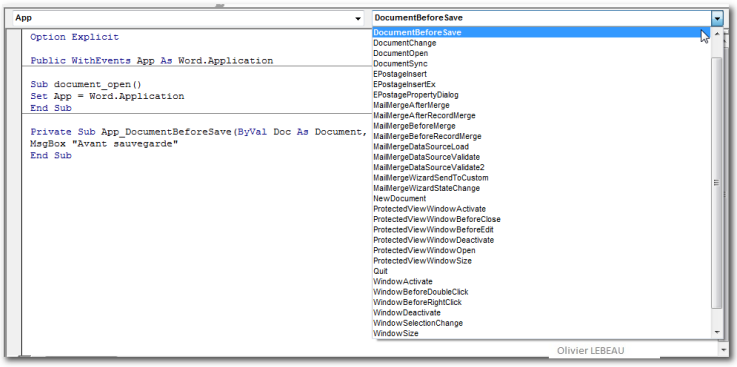
Ce code doit être inséré dans le module ThisDocument.

Ensuite, il faut affecter un objet à cette variable.

Affectation de l'objet application à la variable
```
Sub Document_Open()
Set App = Word.Application
End Sub
```

L'affectation se fera à l'ouverture du document **Document_Open()**.

Il vous reste à choisir dans la liste l'événement que vous souhaitez utiliser.



Liste des événements disponibles

Le code complet :

Le code complet
```
Public WithEvents App As Word.Application

Sub document_open()
Set App = Word.Application
End Sub

Private Sub App_DocumentBeforeSave(ByVal Doc As Document, SaveAsUI As Boolean, Cancel As Boolean)
MsgBox "Avant sauvegarde"
End Sub
```

Ce code va juste avant la sauvegarde d'un document afficher une boîte de dialogue avec le texte "Avant sauvegarde".

II-C-2 - DocumentBeforeClose

Cet événement se produit juste avant la fermeture d'un document.

```
Private Sub appWord_DocumentBeforeClose(ByVal Doc As Document, Cancel As Boolean)
...
End Sub
```

La valeur de l'argument **Cancel** déterminera si le document sera fermé ou non, si elle est **True**, le document restera ouvert.

II-C-3 - DocumentChange

Se produit lors du changement de document actif.

```
Private Sub appWord_DocumentChange()
    MsgBox "Doc Changé"
End Sub
```

III - Les documents

Je pense que l'on peut dire que la collection **Documents** la plus importante de Word, elle est composée d'objets de type **Document**.

Les propriétés et méthodes de la collection sont peu nombreuses.

III-A - Propriétés

La collection possède quatre propriétés, mais une seule nous intéresse vraiment.

III-A-1 - Count

Cette propriété permet de connaître le nombre de documents ouverts. Elle est très simple d'utilisation.

```
Sub NombreDeDocuments()
    Debug.Print Documents.Count
End Sub
```

La propriété Count peut être appliquée sur toutes les collections, elle renvoie le nombre d'éléments ouverts composant cette collection.

III-B - Méthodes

Le nombre de méthodes est quant à lui plus important, nous n'allons en aborder que quelques-unes.

III-B-1 - Open

Cette méthode va ouvrir un document pour l'ajouter à la collection "Documents".

```
Sub OuvrirDocument()
    Documents.Open "C:\Temp\mondocument.dotx"
end Sub
```

III-B-2 - Add

Si vous désirez créer un nouveau document, vous devez ajouter un document à la collection et utiliser la méthode **Add**.

```
Sub AjouterDocument()
    Documents.Add
End Sub
```

Si vous ne passez pas d'argument à la méthode, c'est un nouveau document basé sur le modèle normal qui sera créé. Vous pouvez spécifier un modèle de document pour votre méthode en passant son chemin en argument.

```
Sub AjouterDocument()
Documents.Add "C:\Temp\DVP.Dotm"

End Sub
```

Argument	Description	Valeur
Template:=	Chemin complet du modèle de document	Texte
NewTemplate:=	Détermine si le nouveau document sera un modèle ou un simple document	Booléen
DocumentType:=	Détermine le type du nouveau document	constante WdNewDocumentType
Visible:=	Affiche ou non le document	Booléen

III-B-3 - Save

Comme elle s'applique à la collection, cette méthode permet de sauvegarder tous les documents ouverts.

```
Sub SauvegarderDocuments()
    Documents.Save
End Sub
```


Tous les documents.

III-B-4 - Close

Appliquée à la collection, cette méthode ferme tous les documents ouverts.

```
Sub DocumentsClose()
Documents.Close
End Sub
```

III-C - L'élément "Document"

Comme nous venons de le voir, la collection **Documents** est composée de documents, dans les lignes qui précèdent, nous n'avons traité que de la collection entière. Dans cette partie, nous allons traiter les documents mais en tant qu'élément "**Document**".

ActiveDocument Dans les méthodes et propriétés que nous allons explorer, nous allons très souvent faire allusion à "**ActiveDocument**", ce document est le document actif, plutôt que de faire appel à un document par son nom, nous utiliserons donc **ActiveDocument**.

Comment rendre un document actif : il suffit de faire appel à ce document par son Index ou par son nom. Je vous conseille de faire appel au nom du document plutôt que d'utiliser son index. Le nom ne change pas, alors que son index varie en permanence.

Rendre un document actif

```
Documents("MonDocument.docx").Activate
```

Si vous travaillez avec plusieurs documents, il est alors beaucoup plus simple d'utiliser des objets et de leur affecter un document. De cette manière, vous savez exactement quel document vous utilisez.

Affectation de documents à des variables

```
Sub AffectationDeDocuments()
```

Affectation de documents à des variables

```
Dim objDoc1 As Document
Dim objDoc2 As Document

Set objDoc1 = Documents.Open ("C:\Temp\MonDoc1.docm")
Set objDoc2 = Documents.Open ("C:\Temp\MonDoc2.docx")
...
...


With objDoc1
    .Save
    .Close
End With
With objDoc2
    .Save
    .Close
End With
Set objDoc1 = Nothing
Set objDoc2 = Nothing

End Sub
```

III-C-1 - Méthodes

Les méthodes les plus importantes sont utilisées au niveau de la collection, **Open** et **Add** qui renvoient toutes deux un document.

```
Set objDoc1 = Documents.Open ("C:\Temp\MonDoc1.docm")
```

III-C-1-a - Save

Après avoir ouvert ou créé un document, la première action qui nous vient à l'esprit est la sauvegarde.

Sauvegarde d'un document

```
Sub SauvegarderDoc()
'Sauvegarde du document actif
ActiveDocument.Save

End Sub
```

Si le document a déjà été sauvegardé, il sera simplement sauvé avec le nom qu'il possède déjà. Si il n'a pas encore été sauvegardé, vous recevrez une boîte de dialogue vous proposant un nom de sauvegarde.

III-C-1-b - SaveAs

Avec cette méthode, vous devrez spécifier un nom pour votre document, qu'il possède déjà un nom ou pas. Ce nom, vous pouvez le passer en argument dans votre code.

SaveAs d'un document

```
Sub SauverAvecNom()
ActiveDocument.SaveAs

End Sub
```

Si vous ne spécifiez pas d'argument, le document est sauvegardé avec comme nom "Doc1.docx" dans le répertoire par défaut. Il est préférable de ne pas utiliser cette méthode sans argument.

SaveAs d'un document avec argument

```
Sub SauverAvecNom()
ActiveDocument.SaveAs FileName:="C:\temp\MonDocument.docm"

End Sub
```

III-C-1-c - Goto

Cette méthode vous permet d'atteindre un endroit précis de votre document en y déplaçant le point d'insertion.

Méthode Goto

```
Sub Atteindre()
ActiveDocument.Goto What:=wdGotoPage, Which:=wdGoToAbsolute, Count:=5
End Sub
```

Dans l'exemple ci-dessus, nous passons comme argument à la méthode que nous désirons aller à la cinquième page de notre document.
Dans le tableau ci-dessous, vous trouverez la liste des objets (Argument **What**) que vous pouvez atteindre avec la méthode Goto.

Argument What	Description
wdGoToBookmark	Signet
wdGoToComment	Commentaire
wdGoToEndnote	Note de fin
wdGoToEquation	Équation
wdGoToField	Champ
wdGoToFootnote	Note de bas de page
wdGoToGrammaticalError	Erreur de grammaire
wdGoToGraphic	Graphisme
wdGoToHeading	Titre
wdGoToLine	Ligne
wdGoToObject	Objet
wdGoToPage	Page
wdGoToPercent	Pourcentage
wdGoToProofreadingError	Faute de grammaire ou d'orthographe
wdGoToSection	Section
wdGoToSpellingError	Faute d'orthographe
wdGoToTable	Tableau

Cette deuxième liste va compléter la liste des valeurs de l'argument What. Elle donne les valeurs de l'argument Which.

Argument Which	Description
wdGoToAbsolute	Position absolue
wdGoToFirst	Première instance de l'objet spécifié
wdGoToLast	Dernière instance de l'objet spécifié
wdGoToNext	Prochaine instance de l'objet spécifié
wdGoToPrevious	Instance précédente de l'objet spécifié
wdGoToRelative	Position relative à la position actuelle

Si vous utilisez la méthode Goto pour atteindre une page de votre document, c'est le début de la page qui est atteint.

III-C-1-d - PrintOut

Cette méthode permet d'imprimer le document sur l'imprimante par défaut du système d'exploitation.

Impression

```
Sub ImprimerDocument()
ActiveDocument.PrintOut
End Sub
```

Exercice Ouvrez un document via une boîte de dialogue, imprimez le fichier et fermez-le.

III-C-1-e - Convert

Word 2007

Comme, avec la version 2007, un nouveau format de document est apparu, il est possible d'effectuer une mise à niveau de vos documents .doc avec cette méthode

Conversion de document

```
Sub DocumentUpgrade()
ActiveDocument.Convert

End Sub
```

Cette méthode ne prend pas d'argument.

III-C-2 - Propriétés

III-C-2-a - Saved

Si vous voulez savoir si votre document a été sauvegardé depuis la dernière modification, vous devez interroger sur l'état de cette propriété. Si elle est à **True**, la sauvegarde a bien eu lieu. Cette propriété est en lecture/écriture; en modifiant la valeur de cette propriété, vous pouvez signifier à Word que le document n'a pas été modifié.

Faites un petit essai : créez un document et faite un sauvegarde, ajoutez du texte, utilisez ce code :

```
Sub PasDeMessage()
ActiveDocument.Saved = True

End sub
```

Fermez le document, Word ne vous demande rien et les modifications ne sont pas sauvées.

III-C-2-b - Content

Cette propriété en lecture/écriture renvoie le contenu de votre document.

Essayez avec un document contenant peu de texte :

```
Sub ContenuDuDocument()
Debug.Print ActiveDocument.Content
End Sub
```

III-C-2-c - FullName, Name, Path

Ces trois propriétés permettent de récupérer pour la première, le nom et le chemin complet du document, pour la seconde, le nom du document et pour la dernière, le chemin complet du document.

```
Sub CheminEtNom()
With ActiveDocument
    Debug.Print .Name
    Debug.Print .Path
    Debug.Print .FullName
End With
End Sub
```

Si votre document n'a jamais été sauvegardé, il ne possède pas de path et n'a qu'un nom sans extension.

III-C-2-d - AttachedTemplate

Cette propriété renvoie le modèle de document attaché au document, elle est en lecture/écriture.

```
Sub ModeleAttache()
    Debug.Print ActiveDocument.AttachedTemplate
End Sub
```

III-C-2-e - Paragraphs

Cette propriété représente la collection des paragraphes contenus dans le document.

```
Sub NombreDeParagraphes()
    Debug.Print ActiveDocument.Paragraphs.Count
End Sub
```

Nous verrons cette collection en détails un peu plus tard.

III-C-2-f - Sentences

Cette propriété représente la collection des phrases contenues dans le document.

```
Sub NombreDeParagraphes()
    Debug.Print ActiveDocument.Sentences.Count
End Sub
```

Nous verrons cette collection en détails un peu plus tard.

III-C-2-g - Sections

Cette propriété représente la collection des sections contenues dans le document

```
Sub NombreDeParagraphes()
    Debug.Print ActiveDocument.Sections.Count
End Sub
```

Nous verrons cette collection en détails un peu plus tard.

III-C-2-h - Shapes

Cette propriété représente la collection des **Shapes** contenues dans le document

```
Sub NombreDeParagraphes()
    Debug.Print ActiveDocument.Shapes.Count
End Sub
```

Nous verrons cette collection en détails un peu plus tard.

III-C-2-i - InlineShapes

Cette propriété représente la collection des **InlineShapes** contenues dans le document

```
Sub NombreDeParagraphes()
    Debug.Print ActiveDocument.InlineShapes.Count
End Sub
```

Nous verrons cette collection en détails un peu plus tard.

III-C-3 - Évènements

Les évènements liés aux documents sont plus faciles à mettre en oeuvre que pour l'application.

Nous n'allons en citer que quelques-uns et donner un seul exemple.

Évènements des documents

- Close - Se produit à la fermeture d'un document
- Open - Se produit sur l'ouverture d'un document
- New - Se produit lors de la création d'un nouveau document

```
Sub Document_Open()
...
...
End Sub
```

Ce code sera exécuté lors de chaque ouverture de votre document.

IV - Range et Selection

Avant d'aborder le contenu du document, nous allons voir deux objets : **Range** et **Selection**. Ces deux objets sont fort semblables. L'objet **Range** représente une plage de données alors que l'objet Selection est la **sélection** opérée par le curseur. Le point d'insertion dans votre document est la plus simple expression de votre sélection.
Mais un exemple est bien plus pratique pour la compréhension.

Tapez un texte dans un document, si vous n'avez pas vraiment envie d'écrire, utiliser l'expression suivante :

```
=Rand(10,10)
```

suivi de **Enter** dans votre document, vous verrez qu'il se remplit tout seul d'un texte sans queue ni tête qui est malgré tout utilisable pour nos manipulation.

Sélectionnez une portion de texte à la souris et exécutez ce code :

```
Sub AfficherSelection()
Debug.Print Selection.Text

End Sub
```

Vous venez de découvrir l'objet **Selection**.


*Il ne peut y avoir qu'un seul objet **Selection** !*

Pour imager un objet Range, nous allons procéder différemment, toujours avec le même texte utilisez ce code :

```
Sub AfficherRange()
Debug.Print ActiveDocument.Paragraphs(2).Range.Text

End Sub
```

L'objet **Range** représente la plage de données contenue dans un objet Paragraphe.
Essayez cette fois le même code sans faire appel à l'objet Range.

```
Sub AfficherRange()
Debug.Print ActiveDocument.Paragraphs(2).Text

End Sub
```

Vous recevez un magnifique message : Erreur de compilation !

Il est possible d'obtenir un objet sélection au départ du code, essayez :

```
Sub SelectionnerTexte()
ActiveDocument.Paragraphs(4).Range.Select

End Sub
```

Vous verrez que le quatrième paragraphe de votre document est sélectionné comme si vous l'aviez fait à la souris.

Nous aurons l'occasion de revenir plus en détails sur ces objets. Mais il était nécessaire de les aborder sommairement pour la suite.

Nous venons de voir domment récupérer le texte contenu dans un paragraphe avec un objet **Range** et un objet **Selection**. De la même manière, nous pouvons remplacer le texte de ce paragraphe.

```
ActiveDocument.Paragraphs(2).Text = "Mon Texte"
```

Ou en passant par l'object Selection.

```
ActiveDocument.Paragraphs(4).Range.Select
Selection.Text = "Mon texte"
```

V - Les paragraphes

La collection la plus importante d'un document est celle qui contient les paragraphes "**Paragraphs**". Ce sont eux qui contiennent le texte, la mise en forme, ...

Comme nous travaillons avec une collection, il est possible d'atteindre un membre de cette collection par son Index. Si vous désirez atteindre le quatrième paragraphe de votre document, vous devrez utiliser :

```
ActiveDocument.Paragraphs(4)....
```

Dans ce cas, vous renvoyez un objet **Paragraph** de la collection

V-A - En pratique

V-A-1 - Récupérer l'index d'un paragraphe

Pour récupérer l'index d'un paragraphe, il est nécessaire de ruser.
Dans le code ci-dessous, nous partons du point d'insertion, nous étendons la sélection jusqu'au début du document, il ne nous reste qu'à compter le nombre de paragraphes que la sélection contient.

```
Sub RecuperationIndexPara()

Selection.HomeKey Unit:=wdStory, Extend:=wdExtend
Debug.Print Selection.Paragraphs.Count


End Sub
```

V-A-2 - Les styles et les caractères d'un paragraphe

Comme déjà mentionné, le style ainsi que la police de caractère ne sont pas appliqués aux paragraphes mais aux données qu'ils contiennent.

Appliquer Style

```
Sub AppliquerStyle()
ActiveDocument.Paragraphs(1).Range.Style = "Titre 1"

End Sub
```

Pour appliquer un style, vous avez plusieurs possibilités pour le nom du style. Vous pouvez utiliser le nom qui apparait dans la galerie des styles, dans notre exemple "Titre 1", ou utiliser une constante Word : wdStyleHeading1.

De la même manière nous pouvons récupérer le style appliqué au paragraphe :

Récupérer Style

```
Sub RecupererStyle()
Debug.Print ActiveDocument.Paragraphs(1).Range.Style

End Sub
```

Si vous appliquez le style au paragraphe et pas à son contenu, vous ne rencontrerez pas de problème, mais par habitude travaillez avec l'objet Range.

Dans la même optique, vous pouvez modifier la police de caractère d'un paragraphe sans passer par les styles.

Modifier la police de caractère

```
Sub ModifierPolice()
With ActiveDocument.Paragraphs(3).Range.Font
    .Name = "Arial"
    .Bold = True
    .StrikeThrough = True
End With

End Sub
```

Nous l'avons vu plus tôt, With permet d'appliquer à un même objet plusieurs instructions. **Instruction With ... End With**

V-A-3 - Parcourir les paragraphes d'un document

Pour parcourir les paragraphes d'un document, nous allons avoir besoin d'utiliser une boucle.
Les boucles

Les paragraphes (**Paragraph**) font partie de la collection **Paragraphs**, nous allons utiliser une boucle **For Each ... Next**.

```
Sub ParcourirParagraphes()
'Déclaration de l'objet paragraphe
Dim pAra As Paragraph
'Comme nous allons parcourir la collection, il n'est pas utile d'utiliser
'l'opérateur d'affectation Set
For Each pAra In ActiveDocument.Paragraphs
    Debug.Print Len(pAra.Range.Text)
Next pAra

End Sub
```

La variable pAra a volontairement été écrite avec une majuscule en seconde position. De cette manière, lorsque vous saisissez le nom de la variable en minuscule, le VBE corrige automatiquement et vous êtes certain que le nom de variable a été reconnu.

Un paragraphe peut contenir des phrases **Sentences**. Créez un document vierge, tapez : =rand(1,10) suivi Enter pour ensuite essayer ce code.

Compter les phrases d'un paragraphe

```
Sub CompterPhrases()
Debug.Print ActiveDocument.Paragraphs(1).Range.Sentences.Count
End Sub
```

Vous devriez obtenir **10**. Ce code compte le nombre de phrases contenues dans le premier paragraphe du document.

Malheureusement, les mots ne sont pas des objets en Word, mais malgré tout, il existe une collection Words qui représente les mots contenus dans une portion du document.

```
Sub CompterMots()
Debug.Print ActiveDocument.Paragraphs(1).Range.Words.Count
End Sub
```

Exercice :
Dans ce même morceau de document, vous allez compter le nombre de mots dans chaque phrase du premier paragraphe.
Attention, l'objet Word n'existe pas, vous devrez le déclarer en tant que Variant.

Exercice :
Créez un nouveau document, remplissez le avec : =rand(10,2), supprimez le dernier paragraphe vide
Mettez en gras et double souligné le troisième mot de chaque paragraphe.

V-B - Méthodes

V-B-1 - Add

De la même manière qu'avec les autres collections, la méthode **Add** va ajouter un paragraphe.

```
Sub AjouterParagraph()

ActiveDocument.Paragraphs.Add Range:=Paragraphs(3).Range

End Sub
```

L'argument Range passé à la méthode peut être omis et dans ce cas, le paragraphe est ajouté au point d'insertion.

```
Sub AjouterParagraph()

ActiveDocument.Paragraphs.Add

End Sub
```


Il existe une méthode de l'objet Selection qui produit le même effet :

```
Sub AjouterParagraphe()
Selection.InsertParagraph
End Sub
```

V-B-2 - Space1 - Space15 - Space2

Cette méthode ajoute un interligne Simple, de 1,5 ou double aux paragraphes spécifiés.

La mise en oeuvre est assez simple :

```
ActiveDocument.Paragraphs.Space15
```

Cette méthode peut s'appliquer à la collection ou à un membre de la collection.

```
ActiveDocument.PAragraphs(1).Space2
```

V-B-3 - OutlineDemote - OutlinePromote

Ces deux propriétés sont similaires, seule leur action est différente. Dans le cas de Promote, on va changer le niveau de titre des paragraphes spécifiés pour monter le niveau de titre de 1 et pour Demote, on va diminuer le niveau de titre de 1.

Si vous avez un paragraphe de niveau de titre 2 et que vous lui appliquez la méthode OutlinePromote, vous aurez un paragraphe de niveau de titre 1.
De la même manière, la méthode OutlineDemote va diminuer le niveau de titre de 1, si vous appliquez cette méthode à un paragraphe de titre 2, vous obtiendrez un titre de niveau 3.

Il est beaucoup plus facile d'attribuer un style de titre que de passer par cette méthode.

```
ActiveDocument.Paragraphs(4).OutlineDemote
```

V-C - Propriétés

Les propriétés des objets paragraphes sont plus interressantes que les méthodes.

V-C-1 - Alignment

Cette propriété permet l'alignement du paragraphe, vous avez plus de possibilités qu'avec l'interface graphique.

```
ActiveDocument.Paragraphs(1).Alignment = wdAlignParagraphRight
```

Alignment peut prendre les valeurs ci-dessous :

Valeur	Description
wdAlignParagraphCenter	Aligné au centre.
wdAlignParagraphDistribute	Les caractères du paragraphe sont distribués équitablement pour remplir toute la largeur du paragraphe.
wdAlignParagraphJustify	Totalement justifié.
wdAlignParagraphJustifyHi	Justifié avec un rapport élévé de compression de caractères.
wdAlignParagraphJustifyLow	Justifié avec un faible rapport de compression de caractères.
wdAlignParagraphJustifyMed	Justifié avec un rapport moyen de compression de caractères.
wdAlignParagraphLeft	Aligné à gauche.
wdAlignParagraphRight	Aligné à droite.

V-C-2 - Borders

Ajoute des bordures au paragraphe spécifié. Un exemple sera plus intéressant qu'une description :

Ajouter une bordure

```
Sub AjouterBordure()
'Ajoute les bordures au paragraphe
With ActiveDocument.Paragraphs(6)
    .Borders(wdBorderLeft).Visible = True
    .Borders(wdBorderBottom).Visible = True
    .Borders(wdBorderTop).Visible = True
    .Borders(wdBorderRight).Visible = True
End With
'Modifie l'aspect de la bordure
With ActiveDocument.Paragraphs(6).Borders
    .OutsideLineStyle = wdLineStyleDashDot
End With

End Sub
```

VI - Les tableaux

VI-A - En pratique

Les tableaux font partie de la collection **Tables**. Le plus simple pour les adresser est de passer par leur index.

```
ActiveDocument.Tables(1)
```

Il est possible de parcourir les différentes tables de votre document à l'aide d'une boucle.

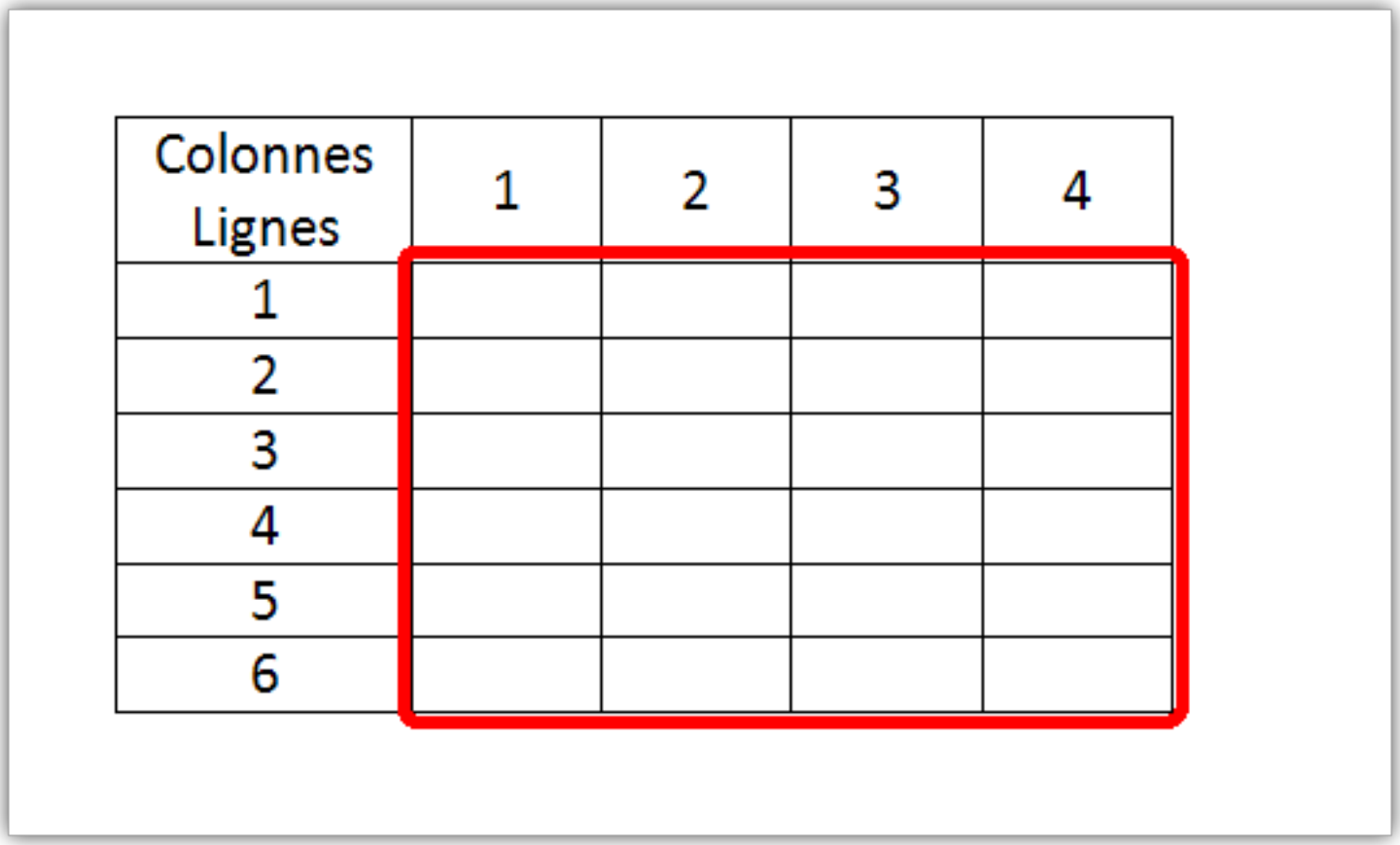
```
Sub ParcourirTables()
Dim oTbl As Table

For Each oTbl In ActiveDocument.Tables
    ...
Next oTbl
End Sub
```

Il n'est pas possible d'affecter un nom à un tableau en Word.

Les tableaux sont constitués de lignes et de colonnes.

```
Sub CompterLignesEtColonnes()
Debug.Print ActiveDocument.Tables(1).Rows.Count 'Nombre de lignes
Debug.Print ActiveDocument.Tables(1).Columns.Count 'Nombre de colonnes
End Sub
```

En plus de l'adressage des tableaux, il est possible d'adresser chaque cellule de chaque tableau pour y parvenir, c'est toujours le même principe, on utilise les coordonnées de la cellule.



Coordonnées dans un tableau

Pour ajouter du texte dans une cellule, vous devez adresser la cellule et ensuite affecter à l'objet Range de la cellule le texte voulu.

```
Sub AjouterTexte()
ActiveDocument.Tables(1).Cell(1,1).Range.Text = "A1"
End Sub
```

Exercice
Récupérez le texte que vous venez d'ajouter dans la cellule avec un Debug.Print.

VI-A-1 - Ajout d'un tableau dans un document

Comme pour beaucoup d'objets, la méthode pour ajouter est **Add**.

Ajouter une table

```
Sub AjouterTableau()
ActiveDocument.Tables.Add Range:=Selection.Range, numrows:=5, numcolumns:=6
End Sub
```

Cette table de 5 lignes et 6 colonnes sera ajoutée à l'emplacement du curseur de la souris.

Exercice
Vous allez sur le document actif créer une table de 3 colonnes et 3 lignes.
Par le code, vous allez créer un nouveau document dans lequel vous allez ajouter une table contenant le même nombre de lignes et de colonnes.

VI-A-2 - Ajouter une ligne ou colonne à un tableau

En Word, un tableau ou table est constitué de lignes et de colonnes, vous avez utilisé l'exercice précédent deux arguments pour ces objets. Vous pouvez récupérer le nombre de lignes et de colonnes de votre tableau en utilisant la propriété **Count**.

Nombre de lignes et de colonnes

```
Sub AfficherLignesEtColonnes()
Dim intR As Integer
Dim intC As Integer
Dim strMessage As String
'Récupération du nombre de lignes et de colonnes
intR = ActiveDocument.Tables(1).Rows.Count
intC = ActiveDocument.Tables(1).Columns.Count
'Élaboration du message par concaténation
strMessage = "Votre tableau contient : " & vbCrLf
strMessage = strMessage & intC & " colonnes " & vbCrLf
strMessage = strMessage & intR & " lignes."
'Affichage du message
MsgBox strMessage

End Sub
```

Pour ajouter une ligne ou une colonne à un tableau, nous allons procéder de la même façon, mais nous allons utiliser la méthode **Add** au lieu de la propriété **Count**.

Ajouter une ligne et une colonne

```
Sub AjouterLigneEtColonne()
Dim oTbl As Table

Set oTbl = ActiveDocument.Tables(1)
oTbl.Rows.Add
oTbl.Columns.Add
Set oTbl = Nothing
End Sub
```

Ce code va simplement ajouter une ligne et une colonne, à la fin de votre tableau pour la ligne et à la droite pour la colonne. Ce serait beaucoup plus intéressant de pouvoir ajouter cette ligne ou cette colonne à un endroit de votre choix.
La méthode **Add** peut prendre un argument pour l'ajout.

Ajouter une ligne et une colonne 2

```
Sub AjouterLigneEtColonne()
Dim oTbl As Table

Set oTbl = ActiveDocument.Tables(1)
```

Ajouter une ligne et une colonne 2
```
oTbl.Rows.Add BeforeRow:=oTbl.Rows(2)
oTbl.Columns.Add BeforeColumn:=oTbl.Columns(2)
Set oTbl = Nothing
End Sub
```

Dans l'exemple ci-dessus, nous ajoutons une ligne avant la seconde et pareil pour la colonne. La seule difficulté est la valeur passée à l'argument, ce n'est pas un entier, mais un objet ligne ou colonne.

VI-A-3 - Contenu d'une cellule

Lorsque vous ajoutez ou utilisez un tableau dans un document, vous pouvez aussi ajouter des données dans les cellules. Pour le faire, la méthode est assez simple, il suffit d'adresser cette cellule et d'affecter les données au Range de la cellule

Ajouter du contenu à une cellule
```
ActiveDocument.Tables(1).Cell(1,1).Range.Text = "Mon Texte"
```

Pour récupérer le contenu d'une cellule, la méthode est la même que pour l'ajout de données.

Récupérer le contenu d'une cellule
```
Debug.Print ActiveDocument.Tables(1).Cell(1,1).Range.Text
```

Nous avons vu comment ajouter un contenu à une cellule et un exercice pour le récupérer. Lors de cette récupération, vous aurez probablement remarqué que le texte récupéré est plus long que le texte ajouté. Ce phénomène est dû aux caractères non imprimables qui délimitent la cellule. Pour remédier à cet inconvénient, nous allons fabriquer une petite fonction utilisant certaines fonctions que nous avons vues dans les chapitres précédents, **http://heureuxoli.developpez.com/office/word/vba05/#L3-A-1** et **http://heureuxoli.developpez.com/office/word/vba05/#L3-A-4**.

Nous allons récupérer le contenu, mesurer sa taille en comptant le nombre de caractères présents et nous allons prendre tous les caractères sans les deux derniers. Nous allons appeler cette fonction **NetText** et lui passer en paramètre le texte contenu dans la cellule qui est à traiter.

Fonction NetText
```
Public Function NetText(stTemp As String) As String
'Nous utilisans les caractères de la chaîne sans les deux derniers
NetText = Left( stTemp , Len(stTemp) - 2 )
End Sub
```

Souvenez-vous de cette fonction, vous en aurez certainement besoin.

L'utilisation de l'ensemble est assez simple :

Utilisation de notre Fonction
```
Debug.Print NetText(ActiveDocument.Tables(1).Cell(1,1).Range.Text)
```

Il est fréquent que l'on doive d'ajouter une ligne à un table pour y injecter des données. Pour y parvenir, nous allons utiliser la méthode Add et ensuite la propriété Count pour obtenir le nombre de ligne, de cette manière, nous pourrons adresser la dernière.

```
Sub AjouterLigneEtDonnees()
```

```
Dim oTbl As Table

Set oTbl = ActiveDocument.Tables(1)
oTbl.Rows.Add
oTbl.Rows(oTbl.Rows.Count).Cell(1).Range.Text = "Dernière Ligne"
Set oTbl = Nothing
End Sub
```

Exercice
Nous avons vu comment ajouter un tableau à un document et comment adresser les cellules de ce tableau.
Vous allez créer un nouveau document, y ajouter un table qui sera une table de multiplication pour les nombres de 1 à 10, et 20 valeurs pour les nombres. Vous ajouterez des titres aux colonnes et aux lignes.

VI-A-4 - Format d'une cellule

En plus d'un contenu, le Range de la cellule possède des attributs permettant de définir le format du texte contenu et de la cellule elle-même.

Vous aurez probablement remarqué que, lorsque vous insérez un tableau dans un document, toutes les lignes sont transparentes. Le code ci-dessous va ajouter une bordure au tableau et définir des lignes pour border les cellules.

```
Sub AjouterContour()
Dim oTbl As Table

Set oTbl = ActiveDocument.Tables(1)

With oTbl.Borders
    .Enable = True
    .InsideLineStyle = wdLineStyleDot
    .OutsideLineStyle = wdLineStyleSingle
    .InsideLineWidth = wdLineWidth050pt
    .OutsideLineWidth = wdLineWidth100pt
End With

End Sub
```

Le texte que vous allez insérer aura le format par défaut des cellules, mais vous pouvez modifier le format du texte contenu dans une cellule.

```
With oTbl.Rows(1).Range.Font
    .Bold = True
    .Underline = wdUnderlineDouble
    .UnderlineColor = wdColorBrightGreen
End With
```

VI-B - Méthodes

VI-B-1 - Add

Nous avons déjà utilisé cette méthode, elle permet l'ajout d'une table à la collection des tables du document.

```
ActiveDocument.Tables.Add Range:=Selection.Range, numrows:=5, numcolumns:=6
```

Les trois arguments ne sont pas facultatifs, Range représente l'endroit où la table sera ajoutée, numrows le nombre de lignes et numcolumns le nombre de colonnes.

VI-B-2 - Delete

Sans aucune surprise, cette méthode va supprimer une table de votre document.

```
ActiveDocument.Tables(1).Delete
```

VI-B-3 - Select

Cette méthode va sélectionner une table de votre document. Vous pourrez ensuite utiliser l'objet **Selection** pour effectuer certaines actions.

```
ActiveDocument.Tables(1).Select
Selection.Copy
Documents.Add
ActiveDocument.Select
Selection.Paste
```

VI-B-4 - Sort

Méthode intéressante, elle permet de trier le contenu d'une table. Il existe trois méthodes de tri pour une table :

- SortAscending
- SortDescending
- Sort

Les deux premières méthodes permettent un tri rapide d'une table. Le tri sera croissant ou décroisant selon que l'on utilise l'un ou l'autre. Leur utilisation est simple et ne comporte pas d'argument.

SortAscending ou SortDescending

```
Sub TriTable()
ActiveDocument.Tables(1).SortDescending

End Sub
```

La première ligne de la table est systématiquement exclue du tri avec ces deux méthodes.

La méthode Sort est un peu plus complexe, mais plus puissante. Vous pouvez avec cette méthode effectuer un tri sur plusieurs colonnes, inclure la première ligne dans votre tri, etc.

VI-C - Propriétés

VI-C-1 - Count

La propriété la plus universelle s'applique aussi aux tables d'un document, elle va renvoyer le nombre d'objets de la collection.

```
debug.Print ActiveDocument.Tables.Count
```

VI-C-2 - AllowAutofit

Si vous voulez redimensionner un tableau en fonction de son contenu, c'est la propriété que vous devez utiliser.

```
ActiveDocument.Tables(1).AllowAutofit
```

VI-C-3 - AllowPageBreaks

Il est très embêtant de voir un tableau coupé en deux par un saut de page. Lorsque cette propriété est sur **False**, votre tableau sera en un seul morceau tant que c'est possible. En effet, si votre tableau fait plusieurs pages, vous n'aurez pas d'autre solution que de le scinder.

```
ActiveDocument.Tables(1).AllowPageBreaks = False
```

VI-C-4 - Borders

Vous aurez probablement remarqué que le tableau que vous insérez via le code n'est qu'une grille sans bordures. Si vous voulez des bordures à votre tableau, vous devez les ajouter.

```
Sub AjoutTableau()
Dim oTbl As Table

Set oTbl = ActiveDocument.Tables.Add(Range:=Selection.Range, NumRows:=5, Numcolumns:=5)
With oTbl.Borders
    .InsideLineStyle = wdLineStyleDouble
    .InsideLineWidth = wdLineWidth025pt
    .OutsideLineStyle = wdLineStyleSingleWavy
End With
Set oTbl = Nothing
End Sub
```

L'utilisation de l'IntelliSense nous facilite la vie pour le choix des constantes à renseigner dans le code.

VI-C-5 - Range

Cette propriété renvoie la partie du document contenue dans un tableau.

```
Sub TablesRange()
Debug.Print ActiveDocument.Tables(2).Range.Text
End Sub
```

Ce code va renvoyer le contenu de toutes les cellules de votre tableau.

VI-C-6 - Tables

Cette propriété peut paraître déplacée, mais une table peut contenir une autre table.

Ajout d'une table dans une cellule

```
Sub AjouterTables()
Dim oTbl As Table
Set oTbl = ActiveDocument.Tables.Add(Range:=Selection.Range, numrows:=2, numcolumns:=2)
oTbl.Cell(1, 1).Select
```

Ajout d'une table dans une cellule

```
Selection.Tables.Add Range:=Selection.Range, numrows:=1, numcolumns:=2
End Sub
```

VII - Les signets

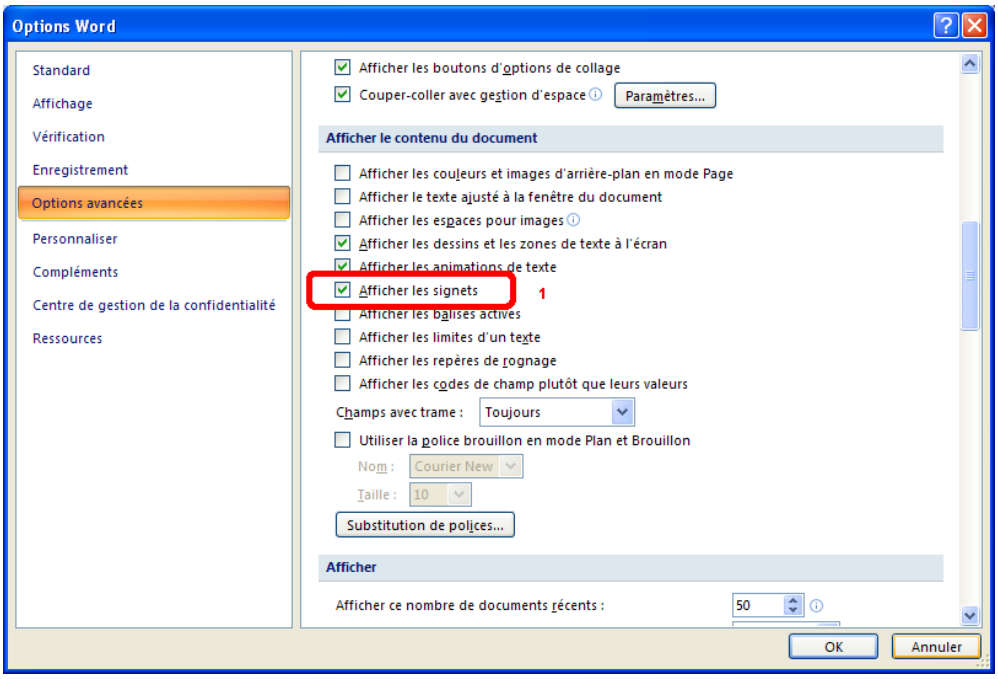
Les signets sont des objets invisibles situés dans votre document et permettant l'accès direct à la position qu'ils occupent. Lorsque vous manipulez le contenu d'un document en VBA, il est bien plus facile d'atteindre un signet que de faire une recherche sur le contenu du document.

Les signets font partie de la collection **Bookmarks** on peut les atteindre par leur nom ou par leur index, comme pour le contenu des tableaux, on y insère des données ou on en récupère le contenu en passant par la propriété **Range**.

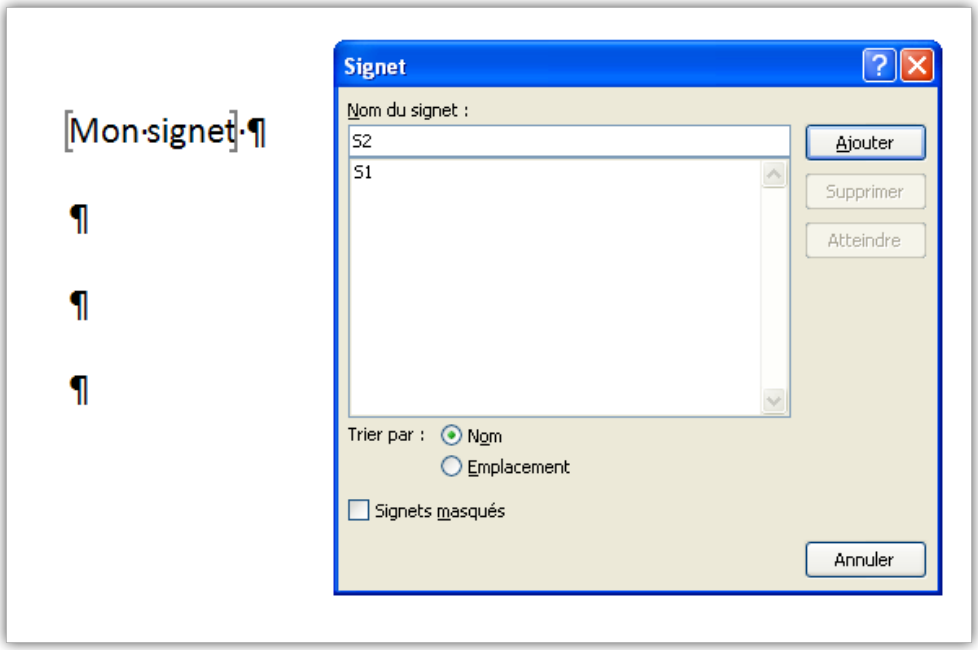
```
ActiveDocument.Bookmarks(1).Range.Text
```

Il existe deux types de signets en VBA, les signets contenant des données et les signets vides qui sont juste un point d'insertion.

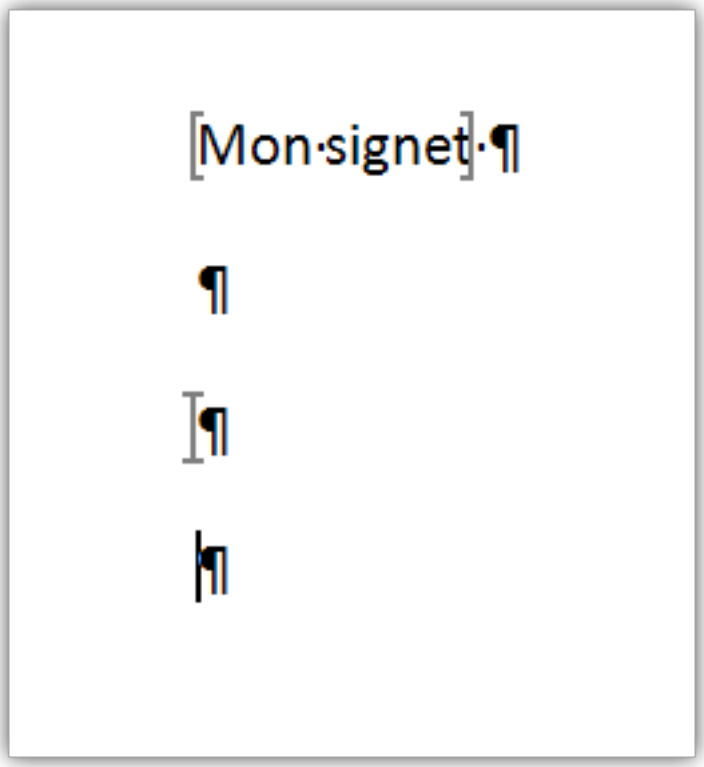
*Pour visualiser les signets présents sur votre document, c'est assez simple, vous devez aller dans les **Options de Word**. **Options Avancées** et dans la partie **Afficher le contenu du document**, vous devez cocher **Afficher les signets**.*



Vous aurez alors deux crochets gris pour symboliser l'emplacement de vos signets.



Créez un document possédant deux signets. Pour le premier, vous écrivez un mot, vous le sélectionnez et ensuite vous insérez un signet sur ce mot. Pour le second, ajoutez quelques paragraphes et ajoutez un signet là ou se trouve votre curseur. Le nom que vous allez donner aux signets n'a pour cet exemple pas d'importance, nous allons les adresser par leur index. Vous devriez obtenir ceci :

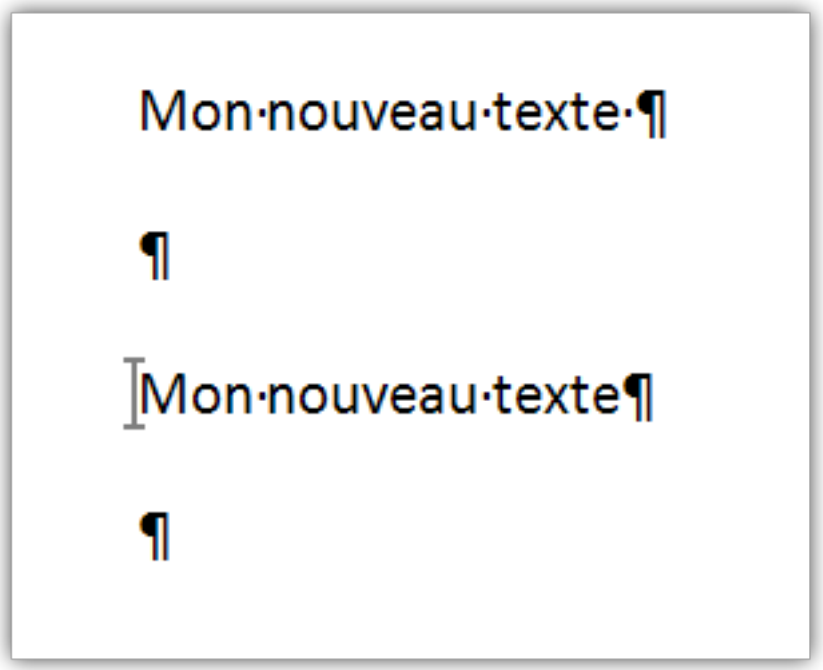


Le premier est un signet normal alors que le second est un point d'insertion. Utilisons un peu de code pour ajouter du texte à nos signets.

```
Sub AjouterTexteSignet()
```

```
Dim stTexte As String
stTexte = "Mon nouveau texte"
ActiveDocument.Bookmarks(2).Range.Text = stTexte
ActiveDocument.Bookmarks(1).Range.Text = stTexte

End Sub
```

Après exécution, nous obtenons ceci :



Le premier signet est détruit et remplacé par le texte, alors que le second existe toujours. Si nous relançons la même procédure, nous obtenons un message d'erreur.

Il est possible de recréer le signet après avoir inséré du texte. Pour y parvenir, nous devons repérer la position de départ du signet et ensuite, connaître la longueur de la chaîne ajoutée au signet.
Nous allons utiliser deux procédures pour cette manipulation, une fonction et une sub.

La fonction

```
Function RemplacerTexteSignet(MyBM As Bookmark, stTexte As String) As Boolean
'Déclaration des variables
'intI pour le début de notre Bookmark
Dim intI As Long
'stBM pour le nom de notre Bookmark
Dim stBM As String
'un objet range pour le range de notre Bookmark
Dim MyRng As Range

'Récupération du nom du signet
stBM = MyBM.Name
'Récupération de la position de départ de notre signet
intI = MyBM.Start
'Affectation du texte à notre Bookmark
MyBM.Range.Text = stTexte
'Affectation de l'objet Range, où la position de départ est
' la même que pour le Bookmark et la fin sera le début augmenté
' de la longueur du texte
Set MyRng = ActiveDocument.Range(Start:=intI, End:=intI + Len(stTexte))
' Crée le Bookmark sur l'objet Range
ActiveDocument.Bookmarks.Add stBM, MyRng
Set MyRng = Nothing
```

La fonction
```
'Affectation de la valeur True à la fonction
RemplacerTexteSignet = True
End Function
```

Nous allons passer en paramètres à cette fonction le signet (MyBM As Bookmark) et le texte à remplacer (stTexte As String).

Il nous reste l'appel de cette fonction :

```
Sub AppelRemplacement()
If RemplacerTexteSignet(ActiveDocument.Bookmarks("S2"), "Le fameux texte à remplacer") Then
    MsgBox "Le remplacement s'est " & vbCrLf _
        & "correctement déroulé !"
End If
End Sub
```

Si le remplacement s'est correctement déroulé, nous affichons un message.

VII-A - Méthodes

VII-A-1 - Exist

Cette méthode permet de savoir si le signet existe dans la collection.

```
If ActiveDocument.Bookmarks.Exists("S1") Then
    ActiveDocument.Bookmarks("S1").Delete
End If
```

Cette méthode renvoie la valeur **True** si le signet existe.

VII-A-2 - Delete

Nous l'avons abordée dans l'exemple précédent. Un signet peut être supprimé par ce code.

```
ActiveDocument.Bookmarks("S1").Delete
```

VII-A-3 - Add

Nous avons déjà fait appel à cette propriété dans notre exemple :

```
ActiveDocument.Bookmarks.Add stBM, MyRng
```

Cette méthode prend deux arguments, le premier (Name:=) est le nom du signet et le second (Range:=), la plage de données qui sera utilisée pour le signet, cette plage peut se résumer à un point d'insertion.

VII-A-4 - Exist

Cette méthode permet de vérifier l'existence d'un signet. Si le signet existe, l'expression renvoie Vrai, Faux dans le cas contraire.

```
ActiveDocument.Bookmarks.Exists("S1")
' Avec S1 comme nom de signet
```

Cette méthode ne s'utilise qu'avec le nom du signet.

VII-B - Propriétés

VII-B-1 - Count

C'est probablement la propriété la plus célèbre, elle permet de connaître le nombre de signets contenu dans la collection des signets (**Bookmarks**).

```
Debug.Print ActiveDocument.Bookmarks.Count
```

VII-B-2 - Name

Cette propriété renvoie le nom du signet, cette propriété est en lecture écriture, nous l'avons déjà utilisée pour notre exemple.

```
Debug.Print ActiveDocument.Bookmarks(1).Name
```

VII-C - Start - End

Ces deux propriétés s'utilisent ensembles. Elles renvoient ou définissent le début et la fin d'un signet

```
ActiveDocument.Paragraphs(6).Range.Select
Selection.Bookmarks.Add Name:="S2"
With ActiveDocument.Bookmarks("S2")
    .Start = ActiveDocument.Paragraphs(6).Range.Characters(3).Start
    .End = ActiveDocument.Paragraphs(6).Range.Characters(6).End
End With
```

Dans cet exemple, nous ajoutons un signet sur le 6^{ième} paragraphe du document, pour ensuite placer ce signet avec pour début le 3^{ième} caractère et la fin au 6^{ième} caractère.

VIII - Solutions des exercices

VIII-A - FileDialog

Dans l'énoncé, nous avons deux arguments à passer à la fonction, le répertoire initial et le titre de la boîte de dialogue. La fonction devra retourner le nom du fichier choisi.

Fonction FileDialog (Solution)

```
Function strFileName( strInitialFolder As String, strTitre As String ) As String
Dim dlg As FileDialog

Set dlg = Application.FileDialog(msoFileDialogFilePicker)
With dlg
    .InitialFileName = strInitialFolder
    .Title = strTitre
```

Fonction FileDialog (Solution)

```
    .Show
End With
strFileName = dlg.SelectedItems(1)
Set dlg = Nothing
End Function
```

L'appel de la fonction

Appel Fonction FileDialog

```
Sub ObtenirFichier()
Debug.Print strFileName("c:\windows\", "www.developpez.com")

End Sub
```

VIII-B - ChangeFileOpenDirectory

Pour y arriver, vous devez déclarer la variable dans le module et pas dans la procédure.

```
Public oldPath As String

'*********************************************************
Sub CahngerDefPath()
oldPath = Application.ChangeFileOpenDirectory
Application.ChangeFileOpenDirectory = "C:\Temp\"
End Sub
'*********************************************************
Sub RestaurerDefPath()
Application.ChangeFileOpenDirectory = oldPath
End Sub
```

VIII-C - Quit

Nous avons abordé comment affecter un objet à une variable. Il existe cinq solutions à cet exercice.

Solution N° 1

```
Dim objWapp As Word.Application

set objWapp = New Word.Application
...
...
objWapp.Quit
Set objWapp = Nothing
```

Solution N° 2

```
Dim objWapp As New Word.Application

...
...
objWapp.Quit
Set objWapp = Nothing
```

Solution N° 3

```
Dim objWapp As Word.Application

set objWapp = Word.Application
...
...
```

Solution N° 3

```
objWapp.Quit
Set objWapp = Nothing
```

Solution N° 4

```
Dim objWapp As Object

set objWapp = CreateObject("Word.Application")
...
...
objWapp.Quit
Set objWapp = Nothing
```

Solution N° 5

```
Dim objWapp As Word.Application

set objWapp = GetObject('"Word.Application")
...
...
objWapp.Quit
Set objWapp = Nothing
```

Dans tous les cas de figure, il faut uiliser **objet.Quit** suivi de **Set objet = Nothing**.

VIII-D - PrintOut

Pour ouvrir le fichier via une boîte de dialogue, nous allons utiliser l'objet FileDialog dont nous avons vu la manipulation. L'étape suivante est l'ouverture du fichier suivie de son impression et finalement sa fermeture.

Exercice PrintOut

```
Sub ImpressionFichier()
Dim objDlg As FileDialog
Dim strNomFichier As String
'Affectation des objets
Set objDlg = Application.FileDialog(msoFileDialogFilePicker)

objDlg.Show
' Test de vérification de fichier choisi
' si pas de fichier choisi, on sort de la procédure
If objDlg.SelectedItems.Count = 0 Then
    Set objDlg = Nothing
    Exit Sub
End If
'Récupération du nom de fichier
strNomFichier = objDlg.SelectedItems(1)
'Ouverture du fichier
Documents.Open strNomFichier
'Impression
ActiveDocument.PrintOut
'Fermeture
ActiveDocument.Close
'libération des objets
Set objDlg = Nothing


End Sub
```

Dans notre code, nous avons introduit une condition supplémentaire, s'il nt a pas de fichier choisi, en faisant ce test, nous sortons de la procédure .

VIII-E - Compter les mots des phrases

Nous allons parcourir la collection des phrases pour compter le nombre de mots contenu dans chaque phrase du premier paragraphe.

```
Sub CompterMots()
Dim wdSentence
' ne pas donner de type à la variable équivaut à la déclarer
' comme Variant
' Faire une boucle sur la collection Sentence et en compter les mots
For Each wdSentence In ActiveDocument.Paragraphs(1).Range.Sentences
    Debug.Print wdSentence.Words.Count
Next wdSentence
End Sub
```


Ne pas déclarer le type de la variable ne doit être utilisé que si ce type n'existe pas.

VIII-F - Mettre troisième mot en gras et double souligné

Comme nous devons parcourir tous les paragraphes de la collection, nous allons déclarer un objet paragraphe et une boucle **For Each ... Next**. Pour chaque paragraphe, nous allons adresser le troisième mot de la collection des mots de ce paragraphe.

Mot trois gras et souligné

```
Sub MettreMot3Gras()
'Déclaration des variable
Dim pAra As Paragraph
'Boucle sur les paragraphes du document
For Each pAra In ActiveDocument.Paragraphs
    With pAra.Range.Words(3).Font
        .Bold = True
        .Underline = wdUnderlineDouble
    End With
Next pAra

End Sub
```

VIII-G - Ajouter un document contenant une table

Pour ajouter une table vous devez spécifier le nombre de lignes et de colonnes que votre table va contenir. Pour y parvenir, nous allons utiliser deux variables qui vont contenir le nombre de lignes (intR) et de colonnes (intC). Pour l'ajout de la nouvelle table, nous utiliserons ces deux variables.

```
Sub AjouterDocEtTable()
Dim intR As Integer
Dim intC As Integer

With ActiveDocument.Tables(1)
    intR = .Rows.Count 'Nombre de lignes
    intC = .Columns.Count 'Nombre de colonnes
End With
'Ajout du nouveau document
Documents.Add
'Ajout d'une table
ActiveDocument.Tables.Add Range:=Selection.Range, numrows:=intR, numcolumns:=intC

End Sub
```

Comme le document ajouté devient le Document Actif, pour ajouter la table, nous utiliserons **ActiveDocument**.

VIII-H - Table de multiplication

Avant de créer notre document, nous devons déterminer combien lignes et de colonnes votre table va comporter. L'énoncé demande 10 nombres, 20 valeurs et un titre par colonne et par ligne. Ce qui va nous donner 11 colonnes et 21 lignes.
Nous allons en profiter pour changer l'apparence de notre table en ajoutant des bordures extérieures et intérieures, les titres des lignes et colonnes seront en gras souligné.

Pour l'ajout d'un nouveau document, nous l'avons déjà vu plus tôt.

```
Dim oDoc As Document

Set oDoc = Application.Documents.Add
```

Pour l'ajout d'une table, vous l'avez déjà fait.

```
Dim oTbl As Table
Dim intR As Integer 'Nombre de lignes
Dim intC As Integer 'Nombre de colonnes


' Ajout de notre table
Set oTbl = oDoc.Tables.Add (Range:=Selection.Range, NumRows:= 21 , NumColumns:=11)
```

Pour les résultats, nous allons utiliser une boucle :

```
For intR = 2 To 21
    For intC = 2 To 11
        oTbl.Cell(intR, intC).Range.Text = (intC - 1) * ( intR - 1)
    Next intC
Next intR
```

Il nous manque les données pour les titres des lignes et colonnes et nous allons en profiter pour mettre notre test en forme.

```
For intR = 2 To 21
    'Données de la première ligne
    oTbl.Cell(intR, 1).Range.Text = intR - 1
    'Mise en forme de chaque cellule
    With oTbl.Cell(intR, 1).Range.Font
        .Bold = True
        .Underline = wdUnderlineDouble
        .UnderlineColor = wdColorBlack
    End With
Next intR
For intC = 2 To 11
    'Données de la première colonnes
    oTbl.Cell(1, intC).Range.Text = intC - 1
    'Mise en forme
    With oTbl.Cell(1, intC).Range.Font
        .Bold = True
        .Underline = wdUnderlineDouble
        .UnderlineColor = wdColorBlack
    End With
Next intC
```

Finalement, il nous reste la mise en forme de la table :

```
With oTbl.Borders
    .Enable = True
    .InsideLineStyle = wdLineStyleDot
    .OutsideLineStyle = wdLineStyleSingle
    .InsideLineWidth = wdLineWidth050pt
    .OutsideLineWidth = wdLineWidth100pt
End With
```

Si nous assemblons tous nos petits morceaux de code, nous obtenons ceci :

Le code complet

```
Sub TableDeMultiplication()
Dim oDoc As Document
Dim oTbl As Table
Dim intR As Integer 'Nombre de lignes
Dim intC As Integer 'Nombre de colonnes

Set oDoc = Application.Documents.Add
' Ajout de notre table
Set oTbl = oDoc.Tables.Add(Range:=Selection.Range, NumRows:=21, NumColumns:=11)
'Les résultats
For intR = 2 To 21
    For intC = 2 To 11
        oTbl.Cell(intR, intC).Range.Text = (intR - 1) * (intC - 1)
    Next intC
Next intR
'Les titres
intR = 0
intC = 0
For intR = 2 To 21
    oTbl.Cell(intR, 1).Range.Text = intR - 1
    With oTbl.Cell(intR, 1).Range.Font
        .Bold = True
        .Underline = wdUnderlineDouble
        .UnderlineColor = wdColorBlack
    End With
Next intR
For intC = 2 To 11
    oTbl.Cell(1, intC).Range.Text = intC - 1
    With oTbl.Cell(1, intC).Range.Font
        .Bold = True
        .Underline = wdUnderlineDouble
        .UnderlineColor = wdColorBlack
    End With
Next intC

'Mise en forme
'Bordure
With oTbl.Borders
    .Enable = True
    .InsideLineStyle = wdLineStyleDot
    .OutsideLineStyle = wdLineStyleSingle
    .InsideLineWidth = wdLineWidth050pt
    .OutsideLineWidth = wdLineWidth100pt
End With



End Sub
```

IX - Liens Utiles

	Titre de l'article
VBA	**Généralités sur le VBA**
	Initiation au VBA d'Outlook

X - Remerciements

Je tiens à remercier pour leurs contributions à ce projet :
- **lorenzole+bo**
- **Philippe JOCHMANS**
- **Caro-Line**
- **Arkham46**
- **Jeannot45**
- **dourouc05**

Un grand merci à **Antoun** pour son aide et ses corrections avisées.