PROGRAMMATION ORIENTÉE OBJET

Le VBA est un langage de Programmation Orientée Objet (POO), c'est-à-dire qu'il utilise différents objets (classeur, feuille, cellules, etc.) auxquels il associe des attributs (taille, couleurs, nom, etc.), des valeurs ("Bonjour", 357, etc.), des événements (activer, fermer, etc.)… Voici donc une présentation de quelques objets avec la programmation de leurs attributs.

DÉFINITIONS

OBJET

Un objet est un élément (classeur, feuille, etc.) dont on peut définir ou modifier les propriétés.

PROCÉDURE

La procédure va accueillir le code de la macro. Elle commence par l'instruction **Sub** et se termine par l'instruction **End Sub**.

MODULE STANDARD

Le module standard est le support permettant d'écrire sa procédure.

PROJET

Un projet VBA est l'ensemble des modules de code VB associé à un classeur.

COLLECTION

La collection permet de travailler sur plusieurs objets.

PRINCIPE DES OBJETS

PROPRIÉTÉS DES OBJETS

Chaque objet a ses propres attributs.

On y fait référence par *Objet.Propriété*

Exemple

```
Range("J6").Font.Size = 24 ' La taille du texte de J6 est de 24 points
MsgBox Range("C2").Interior.Color ' Affiche le code de la couleur du fond
Columns("E:G").ColumnWidth = 28 ' Élargit les colonnes de E à G à 28 points
```

MÉTHODES

Les méthodes sont des procédures ou des fonctions attachées aux objets.

Leur structure est : *objet.méthode argument1,argument2,…*

Exemples

```
Range("B2:D12").Select ' sélection des cellules de B2 à D12
Selection.Name = "Taux" ' Nomme « Taux » la sélection de cellules
ActiveCell.Clear ' Efface le contenu de la cellule active
```

LES OBJETS D'EXCEL

OBJET *WORKBOOKS*

Les objets de la classe *Workbook* sont des classeurs Excel

EXEMPLE D'APPEL D'OBJETS DE LA CLASSE WORKBOOK :

Ouverture du classeur *Compta.xlsx* situé dans le même dossier que le classeur actuel :

```
Workbooks.Open "Compta.xlsx"
```

Sauvegarde le classeur actif sous le nom de *Facture.xlsx* dans le sous- dossier *Clients* :

```
ActiveWorkBook.SaveAs ThisWorkbook.Path & "\Clients\Facture.xlsm"
```

- *ActiveWorkbook* : classeur actuellement actif
- *ThisWorkbook :* classeur à partir duquel est lancée la macro

Active le classeur *Ventes.xlsx* (s'il est déjà ouvert)

```
WorkBooks("Ventes.xlsx").Activate
```

Ferme le classeur *Suivi.xlsx*

```
WorkBooks("Suivi.xlsx").Close
```

Création d'un nouveau classeur qui sera sauvegardé dans le même dossier que le classeur à partir duquel est lancée la macro, sous le nom de *Technique.xlsx* :

```
WorkBooks.Add
ActiveWorkBook.SaveAs ThisWorkbook.Path & "\Technique.xlsm"
```

OBJET *WORKSHEET*

Les objets de la classe *WorkSheet* concernent les feuilles de calcul

✓ *Worksheets* désigne les feuilles de calcul
Sheets désigne n'importe quelle feuille (calcul, graphique, etc.)

EXEMPLE D'UTILISATIONS D'OBJETS DE LA CLASSE WORKSHEET

Active la feuille appelée « Janvier » :

```
WorkSheets("Janvier").Activate
```

Active la troisième feuille du classeur « Gestion.xlsx » :

```
Workbooks("Gestion.xlsx").Sheets(3).Activate
```

Renomme la feuille active « Secteur Nord » :

```
ActiveSheet.Name = "Secteur Nord"
```

Supprime la troisième feuille du classeur :

```
Sheets(3).Delete
```

Affiche le nombre de feuille du classeur

```
MsgBox Sheets.Count
```

Ajoute trois feuilles après la dernière feuille du classeur :

```
Worksheets.Add after:=Sheets(Sheets.Count), Count:=3
```

Copie la feuille 1 après la dernière feuille du classeur

```
Sheets(1).Copy after:=Sheets(Sheets.Count)
```

Copie toutes les feuilles du classeur après la dernière feuille

```
Sheets.Copy after:=Sheets(Sheets.Count)
```

PARCOURIR TOUTES LES FEUILLES D'UN CLASSEUR POUR AFFICHER LEUR NOM

```
Dim ActuFeuille As Worksheet ' Définir la variable ActuFeuille comme feuille
For Each ActuFeuille In Worksheets ' Parcourt toutes les feuilles
    MsgBox ActuFeuille.Name ' Affiche le nom de la feuille courante
Next ActuFeuille
```

LOCALISATION DE CELLULES

L'objet Range désigne une plage de cellules

EXEMPLE D'UTILISATION D'OBJETS DE LA CLASSE RANGE

Range

Stocke le mot « Bonjour » dans la cellule A1.

```
Range("A1").Value = "Bonjour" ' .Value est facultatif
```

Copie dans le presse-papiers le contenu des cellules A2 à A30

```
Range("A2:A30").Copy
```

Colle le contenu du presse-papiers dans la cellule D2 de la feuille « Résumé »

```
Worksheets("Résumé").Activate
Range("D2").Select
ActiveSheet.Paste
```

Compte le nombre de cellules contenues dans la zone nommée « Liste »

```
Range("Liste").Count
```

Cells

La variable Tarif récupère le contenu de la cellule C5 :

```
Tarif = Cells(5,3).Value ' Ici aussi .Value est facultatif
```

Écrit 1000 dans la cellule sélectionnée :

```
ActiveCell = 1000
```

Stocke, dans la variable *Emplacement*, les coordonnées de la cellule située deux lignes au-dessus et sept colonnes à droite de la cellule active :

```
Emplacement = ActiveCell.Offset(-2, 7).Address
```

Efface le contenu — valeurs et mise en forme — des cellules de B5 à D20 :

```
Range(Cells(5, 2), Cells(20, 4)).Clear ' Équivalent à Range("B5:D20").Clear
```

✓ *ClearContents* n'efface que les valeurs
ClearFormats n'efface que la mise en forme

Row et Columns

Row et Columns gèrent les lignes et les colonnes :

Met le texte en rouge dans toutes les cellules de la cinquième colonne :

```
Columns(5).Font.Color = -16776961
```

Remplit le fond des cellules de la première ligne en bleu :

```
Rows(1).Interior.Color = 12611584
```

Pour la zone active (ActiveCell)

Affiche le contenu de la cellule de la septième colonne de la ligne contenant le curseur :

```
MsgBox Cells(ActiveCell.Row, 7)
```

Affiche le contenu de la deuxième cellule de la colonne contenant le curseur :

```
MsgBox Cells(2, ActiveCell.Column)
```

COLLECTION D'OBJETS

Une collection d'objets comprend plusieurs objets. Un classeur Excel contient une collection de feuilles, elles-mêmes contiennent une collection de plages qui contiennent une collection de cellule.

Grâce aux collections, il est possible de travailler sur plusieurs objets en même temps, exemple :

```
MsgBox Workbooks("Secteurs.xlsx").Worksheets("Velay").Range("B10")
```

Affiche le contenu de la cellule B10, de la feuille *Velay* du classeur *Secteurs.xlsx*

Définition d'une collection d'objet avec *With* et *End With*

Afin de ne pas répéter à chaque fois la collection d'objet, il est possible de la définir une fois pour toute. Au lieu de répéter à chaque fois le nom du classeur et de la feuille à utiliser :

```
Workbooks("Voyage.xlsm").Worksheets("Europe").Range("A1") = "France"
Workbooks("Voyage.xlsm").Worksheets("Europe").Range("A2") = "Danemark"
Workbooks("Voyage.xlsm").Worksheets("Europe").Range("A3") = "Espagne"
```

Il est plus simple de définir une seule fois le classeur et la feuille à utiliser :

```
With Workbooks("Voyage.xlsm").Worksheets("Europe")
    .Range("A1") = "France"
    .Range("A2") = "Danemark"
    .Range("A3") = "Espagne"
End With
```

✓ Le point devant chaque objet — ici .Range(…) —indique qu'il se situe dans la collection définie avec *With*.

Objet *Application* (ou *WorkSheetFunction*)

Les objets *Application* ou *WorkSheetFunction* permettent d'utiliser les fonctions de calculs prédéfinies d'Excel. Pour afficher le résultat d'une fonction en VBA sans passer par des cellules, il est nécessaire de définir la fonction en tant qu'objet d'application d'Excel. La structure des fonctions sera Application.*Fonction*

Structure d'appel d'un objet de la classe Application

Application.NomFonction(argument1,argument2,…)

Application peut être remplacé par *WorksheetFunction*, exemple :

```
MsgBox WorksheetFunction.Average(Range("A2:A30"))
```

ou

```
MsgBox Application.Average(Range("A2:A30"))
```

Différence entre *Application* et *WorksheetFunction*

La différence entre *Application* et *WorksheetFunction* provient de la gestion des erreurs. *WorksheetFunction* arrête net le programme en cas d'erreur alors que *Application* permet la poursuite du programme s'il est combiné avec, par exemple, IsError().

Exemple *Application*.Search("z", "abc")

```
MsgBox Application.IsError(Application.Search("z", "abc"))
```

Bien que la valeur « z » ne soit pas trouvée dans « abc », le programme se poursuivra, permettant à l'instruction *IsError()* d'affichera la valeur *Vraie* (car c'est vrai qu'il y a une erreur).

Exemple *WorksheetFunction*.Search("z", "abc")

```
MsgBox Application.IsError(WorksheetFunction.Search("z", "abc"))
```

Comme la valeur « z » n'est pas trouvée dans « abc », le programme s'arrêtera net pour cause d'erreur, malgré l'instruction *IsError()*

Quelques fonctions de base

- **Sum** : somme
- **Average** : moyenne
- **Min** : minimum
- **Max** : maximum
- **Median** : valeur médiane
- **Count** : nombre de cellules contenant une valeur numérique
- **CountA** : nombre de cellules non vides.

Exemple d'utilisation de la fonction *Average* pour la moyenne

Affiche la moyenne des cellules de A2 à A30 :

```
MsgBox Application.Average(Range("A2:A30"))
```

SOMME DES CELLULES AVEC LA FONCTION *SUM*

Somme de cellules

Affiche la somme des cellules C2 :C20

```
MsgBox Application.Sum(Range("C2:C20"))
```

Somme de cellules avec arrondi

Toutefois, il se peut qu'il y ait un problème d'arrondi à cause de la gestion des nombres à virgule flottante par le microprocesseur. Dans ce cas, il est possible de définir un arrondi en utilisant la fonction *Round(*valeur à arrondir*,*nombre de décimales*)* avec sa fonction :

```
MsgBox Round(Application.Sum(Range("C2:C20")), 2)
```

RECHERCHE D'UNE VALEUR DANS DES CELLULES

La fonction *Match* permet de retrouver le numéro de la position relative d'un élément dans une liste.

✓ La fonction *Match* correspond à la fonction *Equiv()* d'Excel.

Structure de la fonction *Match*

Application.Match(Valeur cherchée,zone de recherche,Type de recherche)

Type de recherche :

- 0 valeur exacte,
- 1 plus grande valeur inférieure ou égale à la valeur cherchée
- -1 plus petite valeur supérieure ou égale à la valeur cherchée

✓ Si le type de recherche est 1 ou -1, la liste doit être triée par ordre croissant

Exemple d'utilisation de la fonction *Match*

Stocke dans la variable *Lig* le numéro de la première ligne de la zone allant de A2 à A30 de la feuille de calcul *Cours* contenant le mot « Débutant », stocké dans la variable *ActCh*

```
AcCh="Débutant"
Lig = Application.Match("*" & AcCh & "*", Worksheets("Cours").Range("A2:A30"),0)
MsgBox Lig
```

RECHERCHE DE LA VALEUR CORRESPONDANT À UN NOM

La fonction *Index* affiche la valeur correspondant à une ligne et une colonne d'une zone de cellules. Cette ligne ou cette colonne peut être obtenue à partir d'une fonction *Match*.

✓ *Match* et *Index* s'emploient comme les fonctions *Index()* et *Equiv()* d'Excel

Structure de la fonction *Index*

Application.Index(Zone de recherche,Ligne,Colonne)

Structure des fonctions *Index* et *Match* associées

Application.Index(Zone de résultat,
Application.Match(Valeur cherchée,zone de recherche,Type de recherche, 0), 1)

Exemple d'utilisation des fonctions *Index* et *Match*

Affiche le courriel trouvé dans les cellules F2:F20 à la même ligne où se situe le nom *Nivat* trouvé dans les cellules A2:A20 :

```
Courriel = Application.Index(Range("F2:F20"), _
    Application.Match("Nivat", Range("A2:A20"), 0), 1)
```

- Le 0, à la fin, indique une recherche exacte de la fonction *Match* (permettant de retrouver l'information quel que soit le tri)
- Le 1 à la fin indique que l'index se fait dans la première colonne de la sélection (même si, ici, la sélection n'a qu'une seule colonne). ■

OBJETS D'ACTIVATION OU DE DÉSACTIVATION

La plupart de ces commandes peuvent avoir pour attributs *True* (Activée) ou *False* (désactivée). En général, on désactive un paramètre au début de la macro pour le réactiver à la fin.

MISE À JOUR DE L'AFFICHAGE DE L'ÉCRAN

Désactive la mise à jour de l'écran afin d'éviter son scintillement lors de l'exécution de la procédure et surtout, accélère le déroulement de la macro :

```
Application.ScreenUpdating = False ' Désactive la Mise à jour de l'écran
Macro
Application.ScreenUpdating = True ' Réactive la Mise à jour de l'écran
```

CALCULS AUTOMATIQUE

Désactive les calculs automatiques ce qui accélère le temps d'exécution de la macro, mais il ne faudra pas oublier de réactiver les calculs afin de mettre à jour les résultats :

```
Application.Calculation = xlManual ' Calculs manuels
Macro
Application.Calculation = xlAutomatic ' Calculs automatiques
```

MESSAGE D'ALERTES

Désactive les messages d'alertes qui peuvent interrompre le code pour demander à l'utilisateur de valider, par exemple, une sauvegarde :

```
Application.DisplayAlerts = False
' Désactive les messages d'alertes, de confirmation, etc.
Macro
Application.DisplayAlerts = True
' Désactive les messages d'alertes, de confirmation, etc.
```

✓ Les demandes de sauvegarde seront automatiquement validées par défaut

LIENS ENTRE LES CLASSEURS

Désactive la demande de mise à jour des liens à l'ouverture d'un classeur :

```
Application.AskToUpdateLinks = False ' Confirmation des liens entre classeurs
Macro
Application.AskToUpdateLinks = True ' Confirmation des liens entre classeurs
```

ÉVÉNEMENTS

Évite l'exécution de macros événementielles qui se déclenchent lors d'un événement (ouverture d'un classeur, accès à une cellule, saisie d'une valeur, etc.) :

```
Application.EnableEvents = False
' Désactive les événements qui peuvent se déclencher
' à l'arrivée sur un classeur, une feuille, une cellule etc.
Macro
Application.EnableEvents = True ' Réactive les événements
```

BARRE D'ÉTAT

Permet d'afficher le texte de son choix sur la barre d'état, en bas de la fenêtre puis, à la fin de la macro, réactive les messages par défaut de la barre d'état :

```
Application.StatusBar = "En cours de traitement..."
' Affiche le texte "En cours de traitement..." sur la barre d'état
Macro
Application.StatusBar = False ' Remet la barre d'état par défaut
```

POUR CONNAÎTRE SI UN OBJET ÉVÉNEMENTIEL EST ACTIVÉ OU NON

```
MsgBox Application.ScreenUpdating
```

Affiche **Vrai** (True) si la mise à jour de l'écran est activée sinon, affiche **Faux** (False)

ÉVÉNEMENTS

Les macros peuvent s'exécuter lors d'un événement comme l'ouverture d'un classeur, l'ajout d'une feuille, la sélection de cellules, etc.

PROCÉDURES ÉVÈNEMENTIELLES

Des procédures événementielles permettent la gestion de ces événements :

```
With <objet>
   code de la méthode ou les propriétés de l'objet
End With
```

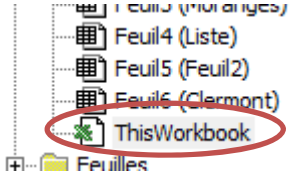
L'instruction *Set* permet de nommer un objet, pour le réutiliser ensuite :

```
Sub NouveauClasseur()
   Dim Classeur As Workbook
   ' Création d'un nouveau classeur :
   Set Classeur = Application.Workbooks.Add
   ' Affectation de noms aux feuilles du classeur :
   With Classeur
     .Worksheets(1).Name = "Janvier"
     .Worksheets(2).Name = "Février"
     .Worksheets(3).Name = "Mars"
   End With
End Sub
```

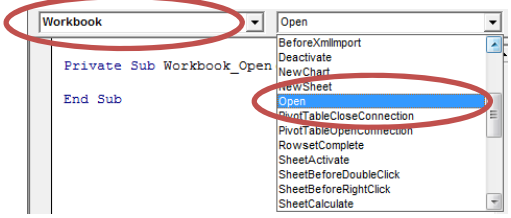
CRÉATION D'UNE PROCÉDURE ÉVÉNEMENTIELLE

Procédure événementielle qui s'exécute à l'ouverture d'un classeur

1. Accéder au code VBA (exemple : **Alt | F11**)
2. Dans le volet d'explorateur de projets, à gauche, cliquer sur « ThisWorkbook »



3. Dans la liste déroulante de gauche, choisir l'élément concerné (exemple : *Workbook* pour un classeur)



4. Dans la liste de droite, choisir l'évènement voulu (exemple : *Open* pour l'ouverture d'un classeur)
5. Éventuellement, effacer les procédures inutiles créées à chaque choix d'un élément dans la liste
6. Saisir le code voulu, exemple :

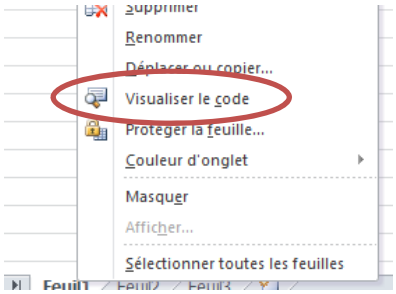
```
Private Sub Workbook_Open()
    Call MessageAccueil ' Lance la macro MessageAccueil
End Sub
```

✓ Voici quelques exemples de procédures gérant les événements à mettre dans le titre de la procédure :

- `Private Sub Workbook_SheetChange(ByVal Sh As Object, ByVal Target As Range)` : déclenchement d'une macro lors de la modification du contenu d'une cellule
  Sh contiendra le nom de l'onglet modifié
  Target sera la cellule modifiée
- `_Open` (comme Workbook_Open) ou `_Beforeclose` : à l'ouverture ou juste avant la fermeture
- `_Activate` ou `_Deactivate` : lors de l'activation ou désactivation du classeur

Procédure événementielle qui s'exécute lors de la modification d'une zone de cellules

1. Clic droit sur l'onglet de la feuille voulue
   et cliquer sur *Visualiser le code*



Il est également possible d'accéder aux procédures d'une feuille en cliquant deux fois sur la feuille voulue dans le volet des projets, à gauche, de la fenêtre du VBA.

2. Dans la liste déroulante de gauche, choisir l'élément concerné
   (exemple : *Worksheet* pour une feuille)
3. Dans la liste de droite, choisir l'évènement voulu
   (exemple : *Change* pour être lancé lors du changement du contenu de cellules).
4. Saisir le code voulu, exemple :

```
Private Sub Worksheet_Change(ByVal Target As Range)
    If Target.Address = Range("A1").Address Then MAJ_Nom
    If Target.Address = Range("A2").Address Then MAJ_Adresse
End Sub
```

✓ Si le contenu de la cellule A1 change, la macro *MAJ_Nom* est exécutée, si c'est le contenu de la cellule A2, c'est la macro *MAJ_Adresse* qui sera exécutée.

Désactiver et réactiver les évènements :

Comme un évènement qui se déclenche lors du changement d'une cellule peut lui-même modifier une cellule, il est parfois conseillé de désactiver le lancement automatique, exemple :

```
Private Sub Workbook_SheetChange(ByVal Sh As Object, ByVal Target As Range)
    Application.EnableEvents = False ' Désactive les évènements
    Call Date_MAJ
    Application.EnableEvents = True ' Réactive les évènements
End Sub
```

Liens sur les événements

Lien vers les différents évènements des classeurs :
silkyroad.developpez.com/VBA/EvenementsClasseur

Site présentant les différents évènements des feuilles de calcul :
silkyroad.developpez.com/VBA/EvenementsFeuille