*Remarque*

Commande vba

**Partie constante de la commande vba**

Explication de la commande

Déclaration des variables et description des types :

Il est toujours préférable d’indiquer la ligne «**Option explicit**» au début de chaque module afin de vous forcer à déclarer les types et donc de vous obliger à toujours savoir quel type de variable vous manipulez.

## Déclarer une variable

En vba il faut distinguer deux grands types d’objet:

### Les objets « Excel » :

**Dim** wb **As Workbook** ‘Déclarer un classeur excel nommé wb

**Dim** ws **As Worksheet** ‘Déclarer une feuille excel nommé ws

**Dim** ch **As Chart** ‘Déclarer un graphique excel nommé ch

**Dim** rg **As Range** ‘Déclarer une plage excel (Exemple A1 :B10)

…

### Les objets « VBA » :

**Dim** i **As Integer** ‘Déclarer un nombre entier

**Dim** s **As String** ‘Déclarer une chaine de caractères

**Dim** d **As Double** ‘Déclarer un nombre décimal

**Dim** da **As Date** ‘Déclarer une date

**Dim** v  **As Variant** ‘Déclarer un variant

*Détaillé dans la section « Les variables de type vba (partie 2) »:*

**Dim i() as integer** ‘Déclarer un tableau d’entier

**Dim s() as string** ‘Déclarer un tableau de chaine de caractères

…

*Dans la suite du cours, on va considérer que chaque type porte le nom donné ci-dessus mais vous auriez pu changer ce nom par exemple* **dim** mon\_classeur **as workbook**

## 

## Attribuer une valeur à une variable précédemment déclarée

Une fois l’objet déclaré, son contenu est toujours vide, vous pouvez alors lui attribuer une valeur :

### Les objets « Excel » :

**Set** wb **=**thisworkbook ‘identifier wb comme étant le classeur contenant la macro

**Set** wb =Activeworkbook ‘identifier wb comme étant le classeur actif

Détaillé dans la section Manipulation des workbooks

**Set** ws = ThisWorkbook.Worksheets("Feuil1") ‘Identifier ws comme étant la feuille nommé “Feuil1” du classeur contenant la macro

*Détaillé dans la section Manipulation des worksheets*

**set rg** =ws.cells(1,1).resize(2,3) ‘Identifier rg comme étant la plage A1:C2 de la feuille ws

*Détaillé dans la section « Manipulation des ranges »*

### Les objets « VBA » :

i=2 ‘Attribuer la valeur 2 à i

s = « Hello world » ‘Attribuer la valeur Hello world à s

d=2.1 ‘Attribuer la valeur 2.1 à d

*en vba le séparateur de décimal est « . » et non pas « , » comme dans la version française d’Excel*

da= Date() ‘Attribuer la date du jour à da

*Date() est une fonction intégrée à vba renvoyant la date du jour*

*Comme vous pouvez le constater, pour attribuer une valeur à un objet Excel, il faut toujours commencer par « set » contrairement aux objets vba.*

## Convertir un type

Prenons :

Dim s as string

dim i as integer, i\_conv as integer

dim s\_date as string, s\_conv as string

dim da\_conv as date

fim d\_conv as double

s\_date ="2016-01-01"

s= «2»

i=5

Il existe 4 fonctions principales permettant de convertir un type en un autre type :

- i\_conv =**CInt(**s) convertir un type en integer

- d\_conv =**CDbl**(s) convertir un type en double

- s\_conv =**CStr**(i) convertir un type en string

- da\_conv =**CDate**(s\_date) convertir un type en date

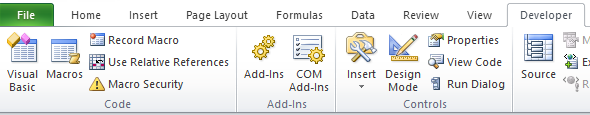
*Bien entendu, si s= « Hello world », la ligne Cint(s) va générer un bug car l’interpréteur ne comprendra pas ce que vous voulez convertir.*

Environnement et debug

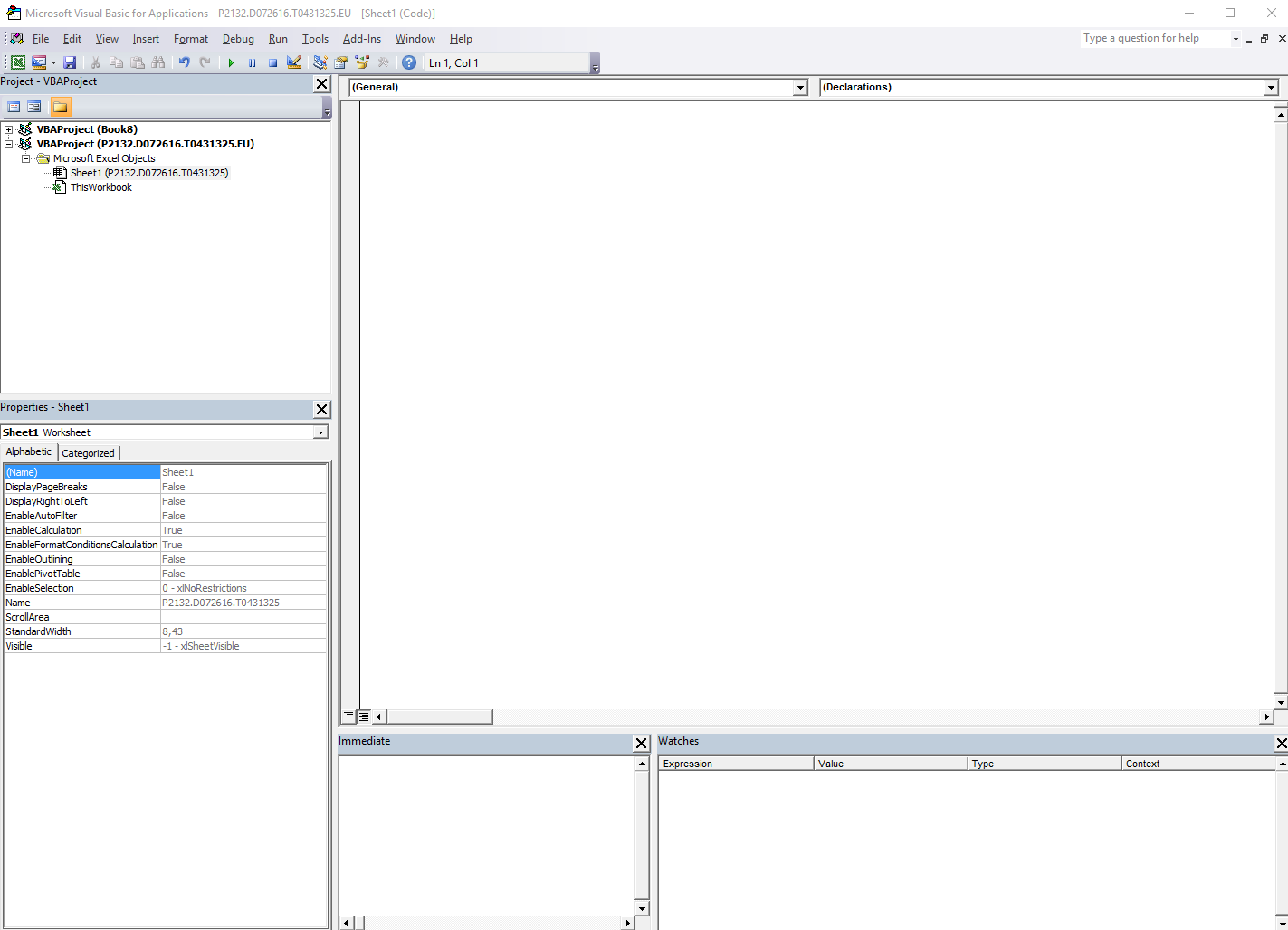
En vba certains outils sont très pratiques notamment pour débuguer. Dans cette partie nous allons décrire l’environnement VBA.

## Ouvrir visual basic :

* ALT F11
* Ou Developer -> Visual Basic



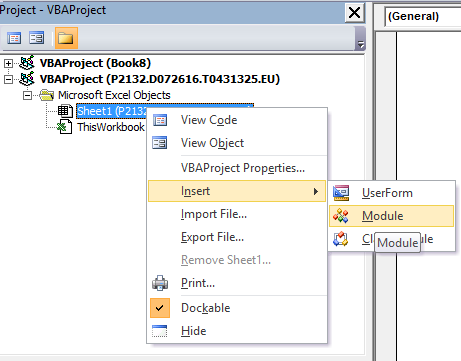
Vous verrez la fenêtre suivante apparaitre :



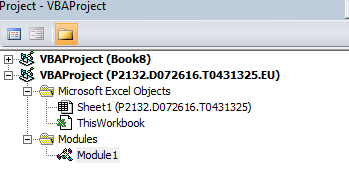
## 

## Ecrire un programme :

Tout d’abord vous devez insérer un module (Clic droit dans la zone ci-dessous -> insert -> Module)



Double clic sur Module 1, une page blanche apparait, c’est là que vous devez taper votre code (i.e. vos sub et vos fonctions).



## Sub et fonction:

### Déclarer un Sub ou une fonction

**Pour un sub :**

**Sub** nom\_de\_la\_sub**(**nom\_de\_la\_variable1 As Double**,** nom\_de\_la\_variable2 As range,…**)**

…

**End Sub**

**Pour une fonction :**

**Function** nom\_de\_la\_fonction**(**nom\_de\_la\_variable1 As Double**,** nom\_de\_la\_variable2 As range,…**) As** type\_retourné

…

**nom\_de\_la\_fonction=**valeur à retourner

**End Function**

### Différence entre un Sub et une fonction

La différence entre un sub et une fonction est simple :

* Un sub ne renvoie jamais de valeur
* Une fonction renvoie toujours une valeur

*Pour lancer votre projet vous aurez besoin d’au moins un sub ne comportant pas d’argument, généralement on l’appelle* main()

Prenons l’exemple d’un programme ayant pour but d’afficher la somme de deux doubles dans la cellule A1 de la feuille « Sheet1 » du classeur contenant la macro. Dans les deux cas, on appelle la sub main pour lancer le programme :

**Methode 1 : En utilisant un sub**

Sub mon\_addition(d\_chiffre1 As Double, d\_chiffre2 As Double)

ThisWorkbook.Worksheets("Sheet1").Cells(1, 1) = d\_chiffre1 + d\_chiffre2

End Sub

Sub main()

Call mon\_addition(2.5, 3.2)

End Sub

**Methode 2 : En utilisant une fonction**

Function mon\_addition(d\_chiffre1 As Double, d\_chiffre2 As Double) **As** Double

mon\_addition = d\_chiffre1 + d\_chiffre2

End Function

Sub main()

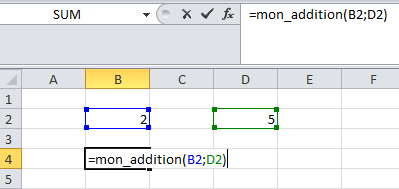
resultat\_de\_la\_fonction = mon\_addition(2.5, 7.2)

ThisWorkbook.Worksheets("Sheet1").Cells(1, 1) = resultat\_de\_la\_fonction

End Sub

### Utiliser une fonction VBA sur Excel

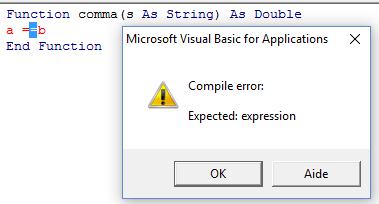
Dans l’exemple précédent, la fonction mon addition permet de prendre en argument deux valeurs et retourne leur somme. Il est possible d’utiliser cette fonction directement dans Excel de la même manière qu’une fonction Excel classique.



*Si votre fonction renvoie un résultat qui sera contenu dans plus d’une cellule, dans ce cas vous devez sélectionner le range qui contiendra le résultat, écrire votre fonction puis à la place de presser ENTER directement, presser CTRL+SHIFT+ENTER*

### Ecrire une ligne d’un programme :

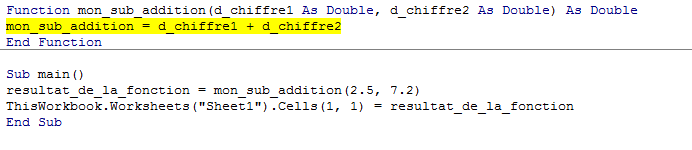
L’avantage de VBA est que vous verrez tout de suite si votre ligne de code est correcte ou non en termes de syntaxe ainsi, si j’écris une commande fausse et que je passe à la ligne, un message d’erreur s’affiche et la ligne apparait en rouge. Vous ne pourrez pas lancer votre programme tant que des lignes resteront rouges.



### Lancer un programme :

Dans l’exemple précédent, pour lancer la fonction main, il faut placer le curseur quelque part dans le main et presser **F5** ou cliquer sur la **flèche verte** 

Vous pouvez également exécuter le programme ligne à ligne en pressant **F8** (ce qui peut s’avérer très utile pour le debug) la ligne sur laquelle le programme est arrêté apparait alors en jaune :



### En cas de bug :

Malgré le fait que votre syntaxe est bonne, il se peut que votre code bug pour de multiple raison au moment où vous lancer votre programme. Dans ce cas vous verrez la fenêtre suivante :

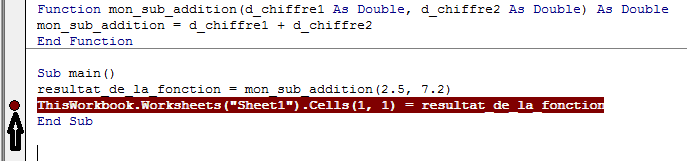
### 

Ici l’indice sur l’erreur est «Type mismatch». En cliquant sur debug, vous verrez apparaitre en jaune la ligne qui a bugué. En cliquant sur End la programme s’arrêtera complètement.

### Utiliser les points d’arrêt :

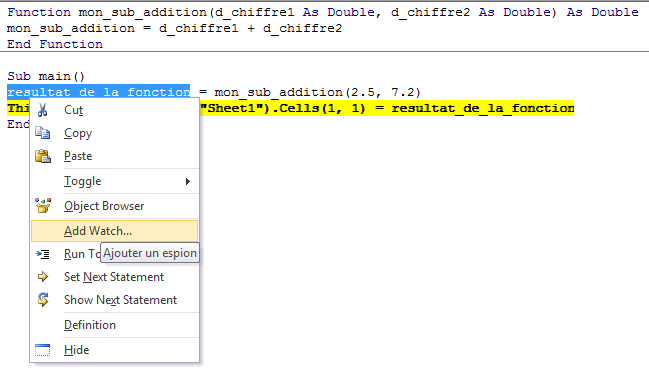
Parfois, vous souhaiterez vous arrêter à un endroit particulier de votre programme (afin de vérifier l’état de vos variables par exemple) mais si votre programme est très long, le bouton F8 n’est pas adapté. Dans ce cas, on utilise un point d’arrêt.

Pour cela, il suffit de cliquer dans la marge à l’endroit où vous souhaitez que le programme s’arrête, un point rouge apparait, vous pouvez alors lancer votre programme avec F5

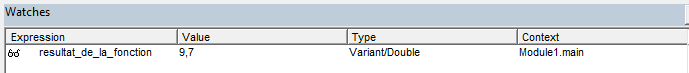


### Utiliser les espions :

Les espions permettent de connaitre à tout moment le contenu et le type d’une variable (à utiliser quand votre programme tourne mais que vous ne comprenez pas pourquoi il ne fait pas ce que vous voulez). Pour ajouter un espion, il faut sélectionner la variable souhaitée puis clic droit « Add watch » ou en français « ajouter un espion » puis cliquer sur « OK » :

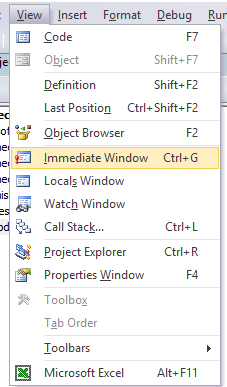


Vous verrez alors apparaitre en bas à droite le nom de la variable, sa valeur ainsi que son type :



### Utiliser la fenêtre d’exécution:

Commençons par afficher la fenêtre d’exécution



Cette dernière permet d’afficher des valeurs lors de l’exécution grâce à la ligne de commande **Debug.print** mavaleur. Un exemple sera plus parlant. Prenons le programme suivant :

Sub mon\_sub()

Dim i As Integer

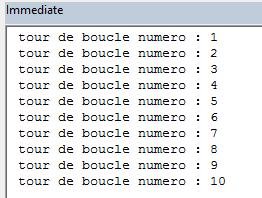
For i = 1 To 10

Debug.Print "tour de boucle numero : " & i

Next i

End Sub

En lançant le programme, la fenêtre d’exécution va alors afficher les différentes valeurs de i :

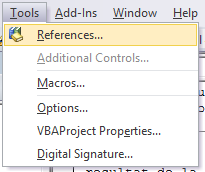


Exercice1

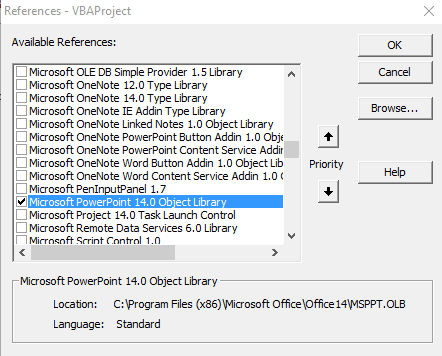
Exercice2

### Utiliser les références :

En vba, vous pouvez avoir accès à des fonctions supplémentaires grâce aux références. Par exemple, ajouter **un package de fonctions** permettant de manipuler Powerpoint. Pour cela il suffit de cliquer sur Tools puis références.



Une liste apparait, cocher la référence souhaitée puis OK

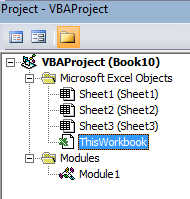


Nous utiliserons par la suite plusieurs références de ce type.

### Lancer une macro à l’ouverture du fichier Excel :

Parfois, il peut être utile de lancer une macro dès l’ouverture d’un fichier Excel pour mettre à jour des formules par exemple. Pour cela, il suffit de :

* Double cliquer sur Thisworkbook



* Dérouler la liste « Generale » et choisir workbook



* Les lignes suivantes apparaissent :

Private Sub Workbook\_Open()

End Sub

Vous pouvez coder votre programme dans ce sub (ou appeler des sub et des fonctions) et celui-ci sera lancé à la prochaine ouverture du fichier.

## Programmation basique :

Dans cette partie nous allons voir les différentes lignes de programmation basique que l’on retrouve dans la plupart des langages informatiques.

### Les opérateurs et comparateurs :

|  |  |
| --- | --- |
| Addition | + |
| Soustraction | - |
| Multiplication | \* |
| Division | / |
| Modulo | Mod |
| Attribuer valeur | = |
| comparer deux valeurs pour vérifier l'égalité | = |
| comparer deux valeurs pour vérifier qu'elles sont différentes | <> |
| comparer deux valeurs pour vérifier que l'une est supérieure à l'autre | > OU < |

Une condition peut inclure plusieurs conditions en utilisant les opérateurs AND et OR. Par exemple valeur1>valeur2 OR valeur1 mod 2 =0 est une condition.

### if :

**If** condition **Then** instruction

OU

**If** condition **Then**

instruction

**End if**

### if/else :

**If** condition **Then**

Instruction1

**Else**

Instrunction2

**End if**

### if/elseif/else :

**If** condition1 **Then**

Instruction1

**ElseIf** condition2 **Then**

Instruction2

**Else**

Instruction3

**End If**

### La condition Select Case:

Case ressemble à un if/elseif/else mais peut parfois être plus commode à rédiger :

**Select Case** variable\_à\_comparer

**Case** valeur1

Instruction1

**Case** valeur2

Instruction2

**Case Else**

Instruction3

**End Select**

**Si la** variable\_à\_comparer est égal à valeur1 c’est l’instruction1 qui s’exécute

**Si la** variable\_à\_comparer est égal à valeur2 c’est l’instruction2 qui s’exécute

Dans tous les autre cas, c’est l’instruction 3 qui s’exécute

### La boucle for:

**For** i = debut **To** fin **Step** iteration

instruction

**Next** i

Par exemple :

**For** i = 0 **To** 10 **Step** 2

Debug.print i

**Next** i

Va afficher 0,2,4,6,8,10

*Si vous ne mettez pas l’instruction* **Step** iteration*, l’itération sera de 1 par défaut*

### La boucle while:

**While** condition

instruction

**Wend**

Par exemple:

i=0

while i<10

debug.print i

i=i+2

wend

Va afficher 0,2,4,6,8,10

### Les balises:

Les balises permettent de forcer le programme à se déplacer directement (en sautant des lignes ou en revenant en arrière) dans une zone définie du programme. Pour cela il suffit d’écrire dans le programme à une ligne donnée ma\_balise**:** puis à un autre endroit du programme **Goto** ma\_balise.

Nous utiliserons cette instruction par la suite notamment pour la gestion des erreurs.

Exercice3

Exercice4

Manipulation des variables

## Les variables de type vba (partie 1):

### Manipulation des doubles:

-Arrondir d\_variable avec n chiffre après la virgule : d\_variable=**Round(**d\_variable**,**n**)**

### Manipulation des strings:

-concaténer deux string : s\_variable=s\_variable1 **&** s\_variable2

-Récupérer les n premiers caractères en partant de la gauche d’un string: s\_variablecut=**Left(**s\_variable,n)

-Récupérer les n premiers caractères en partant de la droite d’un string: s\_variablecut=**Right**(s\_variable,n)

-Récupérer les n caractères en partant du caractère numéro i d’un string: s\_variablecut=**Mid**(s\_variable,i,n)

-Récupérer la longueur d’un string s\_variable : i\_len=**len(**s\_variable)

-Passer a la ligne : vbcrlf

### Manipulation des dates:

-Attribuer une date à une variable : da\_variable=CDate("2016-05-01")

-Attribuer une date à une variable : da\_variable=**DateSerial**("2016", "05", "01")

-Récupérer l’année d’une variable date da\_variable existante : i\_year=**Year(**da\_variable)

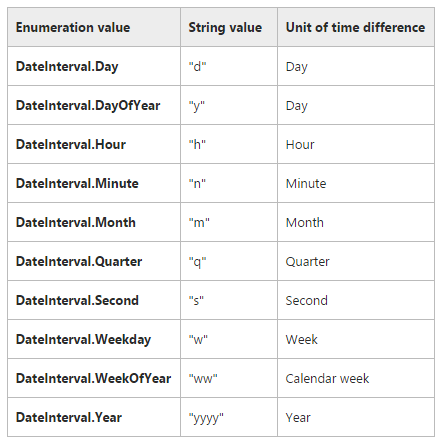
-Récupérer le mois d’une variable date da\_variable existante : i\_month=**Month(**da\_variable)

-Récupérer le jour d’une variable date da\_variable existante : i\_day=**Day(**da\_variable)

-Récupérer le jour courant : da\_today=**Date()**

-Récupérer le jour courant ainsi que l’heure courante : da\_todaytimestamp=**Now()**

-Récupérer la différence entre deux dates en nombre de jours :  
i\_nbday=**DateDiff("**d", da\_variable1, da\_variable2)



Exercice5

## Les variables de type Excel :

### Manipulation des cellules:

* Une cellule est toujours contenue dans une worksheet elle-même contenue dans un workbook.
* Une cellule contrairement aux variables de type vba n’est pas définie uniquement par la valeur qu’elle contient mais également par son format, la formule qu’elle contient, etc …

On aura donc une commande du type :

**Classeur\_contenant\_la\_cellule.Feuille\_contenant\_la\_cellule.cells(numero\_de\_ligne,numero\_de\_colonne).ce\_que\_l’on\_veut\_modifier**

Par exemple :

thisworkbook.worksheets(1).cells(1,1).value va pointer sur **la valeur** de **la cellule A1** de **la feuille numéro 1** du **classeur contenant la macro**

**Remplir des cellules**

-Attribuer une valeur contenue dans s\_variable à une cellule: thisworkbook.worksheets(1).cells(1,1).value=s\_variable

-Attribuer la valeur contenue dans une cellule à une variable s\_variable:

s\_variable =thisworkbook.worksheets(1).cells(1,1).value

*Attention au type de la variable, par exemple, si la cellule contient « Hello » et que s\_variable est déclarée en integer la ligne* s\_variable =thisworkbook.worksheets(1).cells(1,1).value *va planter.*

-Récupérer le numéro de la dernière ligne contenant une valeur dans la colonne n :

i\_nbrows = thisworkbook.worksheets(1**).Cells(Rows.Count,** n**).End(xlUp).Row**

-Récupérer le numéro de la dernière colonne contenant une valeur sur la ligne n :

I\_nbcolumns = thisworkbook.worksheets(1**).Cells(**n, **Columns.Count).End(xlToLeft).Column**

-Permet de pointer sur la cellule décalée de m lignes et n colonnes :

thisworkbook.worksheets(1).cells(2,2).**Offset**(m,n)

par exemple :

thisworkbook.worksheets(1).cells(2,2).**Offset(**-1,-1)

pointe sur A1

**Copier/coller/supprimer des cellules**

-Supprimer le contenu d’une cellule :

thisworkbook.worksheets(1).cells(1,1)**.ClearContents**

-Supprimer le contenu et le format d’une cellule :

thisworkbook.worksheets(1).cells(1,1)**.Clear**

-Copier une cellule:

thisworkbook.worksheets(1).cells(1,1)**.Copy**

-Coller une cellule :

thisworkbook.worksheets(1).cells(2,1**).Paste**

*La valeur copiée à la ligne précédente ainsi que son format sont alors collée dans la cellule A2. Il est nécessaire d’instruire une ligne de copy avant le paste sinon vous ne collez rien …*

-Copier/Coller une cellule :

thisworkbook.worksheets(1).cells(1,1**).Copy Destination:=** thisworkbook.worksheets(1).cells(2,1)

-Copier la ligne n:

thisworkbook.worksheets(1).rows(n)**.copy**

-Copier la colonne n:

thisworkbook.worksheets(1).columns(n)**.copy**

*La plupart des fonctions s’appliquant à cells peuvent s’appliquer directement à rows ou columns*

**Modifier le format et les formules des cellules**

-Modifier le format de la cellule en format date de la forme 31/12/2016:

thisworkbook.worksheets(1).cells(1,1)**.NumberFormat** = "dd/mm/yyyy”

-Modifier le format de la cellule en format décimale avec 3 chiffres après la virgule :

thisworkbook.worksheets(1).cells(1,1)**.NumberFormat** = "0.000”

-Modifier la couleur de fond de la cellule en vert:

thisworkbook.worksheets(1).cells(1,1)**.Interior.Color** = 5296274

-Mettre la police de la cellule en gras:

thisworkbook.worksheets(1).cells(1,1**).Font.Bold** = True

-Appliquer une formule à la cellule (prenons l’exemple du sommeprod):

thisworkbook.worksheets(1).cells(1,1**).Formula = "=**SUMPRODUCT(C24:G24,C25:G25)"

*Vous devez impérativement utiliser la notation anglaise même si vous utilisez une version française d’Excel autrement dit, le nom de la fonction en anglais et des « , » pour séparer les paramètres à la place des « ; »*

### La commande with:

La commande with permet d’économiser quelques caractères dans le code. Prenons l’exemple suivant :

On veut modifier la cellule A1 pour avoir un format "0,000”, un fond vert et du gras.

**Sans le with**

Il suffit de reprendre les lignes précédentes :

thisworkbook.worksheets(1).cells(1,1).NumberFormat = "0,000”

thisworkbook.worksheets(1).cells(1,1).Interior.Color = 5296274

thisworkbook.worksheets(1).cells(1,1).Font.Bold = True

**Avec le with**

Cela donne :

**With** ThisWorkbook.Worksheets(1).Cells(1, 1)

.NumberFormat = "0,000"

.Interior.Color = 5296274

.Font.Bold = True

**End With**

Exercice 6

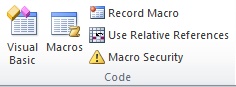
### L’enregistreur de macro :

Il est temps de vous parler d’un outil puissant de VBA à manipuler avec précaution : **l’enregistreur de macro.**

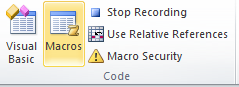
En réalité, il n’est pas nécessaire de connaitre par cœur les lignes décrites précédemment, il suffit de les comprendre et surtout de comprendre l’architecture de la ligne.

Reprenons l’exemple précédent consistant à modifier la cellule A1 pour avoir un format "0.000”, un fond vert et du gras.

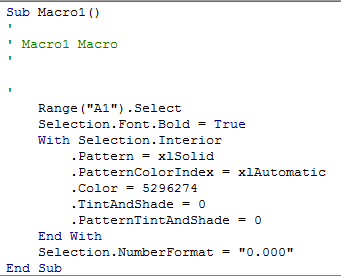
Placer vous sur votre feuille Excel, onglet **Developer**-> **Record Macro**



Une fois que vous avez cliqué sur Record macro, utiliser Excel pour faire vos modifications sur la cellule A1.

Une fois les modifications effectuées cliquer sur **Stop Recording**

Dans l’interface vba un nouveau module a été créé, à chaque instruction sur Excel, des lignes ont été générées. Cela donne :



Vous avez alors deux solutions pour intégrer ce bout de code dans votre programme global :

**Mauvaise pratique :**

Il suffit de copier/coller directement toutes les lignes contenues dans le sub dans votre programme.

**Le problème de cette méthode :**

-Le code peut potentiellement bugger à tout moment

-Si le code bug vous ne serez pas en mesure d’en connaitre la cause car vous n’avez pas compris ce que vous avez codé

-l’enregistreur de macro génère de nombreuses lignes inutiles qui vont polluer votre code, le faire bugger et le rendre illisible

**Bref cette méthode est à bannir.**

**Bonne pratique :**

Vous devez :

-Comprendre chaque ligne (ou presque) générée par l’enregistreur

-Remplacer tous les **Select** et **Selection**, je n’ai pas encore parlé de cette commande, elle est à bannir car génère trop de bug (selon moi)

**Exemple de méthode pour arriver à cleaner le code généré par l’enregistreur :**

* **Code brut :**

Sub Macro1()

'

' Macro1 Macro

'

'

Range("A1").Select

Selection.Font.Bold = True

With Selection.Interior

.Pattern = xlSolid

.PatternColorIndex = xlAutomatic

.Color = 5296274

.TintAndShade = 0

.PatternTintAndShade = 0

End With

Selection.NumberFormat = "0.000"

End Sub

* 1 – On supprime les select pour revenir à une écriture que nous connaissons déjà

Sub Macro1()

'

' Macro1 Macro

'

'

‘ Range("A1").Select

‘ Selection.Font.Bold = True

**Thisworkbook.worksheets(1).cells(1,1).**Font.Bold = True

‘With Selection.Interior

With **Thisworkbook.worksheets(1).cells(1,1).**.Interior

.Pattern = xlSolid

.PatternColorIndex = xlAutomatic

.Color = 5296274

.TintAndShade = 0

.PatternTintAndShade = 0

End With

**Thisworkbook.worksheets(1).cells(1,1).**.NumberFormat = "0.000"

End Sub

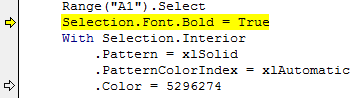
*Les lignes grises sont des lignes supprimées*

*Le but de cette étape est de s’assurer de toujours pointer sur la cellule désirée en utilisant une écriture propre. Dans notre cas, la ligne initiale* Range("A1").Select *pointe sur la cellule A1 sans préciser le nom du classeur et de la feuille. Typiquement cette ligne n’est pas propre et peut pointer sur n’importe quelle cellule A1 du classeur*

Vérifier que le code tourne toujours et fait ce que vous voulez.

* 2 – Comprendre le rôle de chaque ligne et supprimer les lignes inutiles

Pour cela, vous pouvez exécuter le code ligne à ligne (F8 pour rappel) et regarder ce qui se passe à chaque ligne sur la feuille Excel. Vous pouvez également sauter des lignes en faisant glisser la flèche jaune :



Enfin, vous pouvez chercher le rôle de la commande sur internet.

En faisant ce travail, j’obtiens :

Sub Macro1()

**Thisworkbook.worksheets(1).cells(1,1).**Font.Bold = True **Mettre la cellule en gras**

With Selection.Interior

With **Thisworkbook.worksheets(1).cells(1,1).**.Interior

.Pattern = xlSolid

.PatternColorIndex = xlAutomatic

.Color = 5296274 **Changer la couleur du fond en vert**

.TintAndShade = 0

.PatternTintAndShade = 0

End With

**Thisworkbook.worksheets(1).cells(1,1).**.NumberFormat = "0.000" **Changer le format en décimal**

End Sub

Le code final est donc :

**Thisworkbook.worksheets(1).cells(1,1).**Font.Bold = True **Mettre la cellule en gras**

**Thisworkbook.worksheets(1).cells(1,1).**.Interior.Color = 5296274  **Couleur du fond en vert**

**Thisworkbook.worksheets(1).cells(1,1).**.NumberFormat = "0.000" **Changer le format en décimal**

***En fait l’enregistreur de macro ne permet pas d’écrire des programmes automatiquement, il permet simplement de comprendre comment accéder à une propriété d’un objet***

Exercice 7

### Manipulation des classeurs (Workbook):

-Ouvrir un classeur nommé classeur1.xlsx et situé dans C:\Users\desk\Documents:

path = « C:\Users\desk\Documents\ classeur1.xlsx »

**Workbooks.Open Filename:=** “ C:\Users\desk\Documents\ classeur1.xlsx »

-Laisser le choix à l’utilisateur d’ouvrir le classeur qu’il souhaite grâce à une boite de dialogue:

**Workbooks.Open Filename:= application.getopenfilename()**

-Créer un nouveau classeur:

**Workbooks.add**

-Une fois le classeur ouvert ou créé, **vous devez impérativement à la ligne suivante l’associer à une variable** Pour cela il faut utiliser la ligne suivante:

**Set** wb **= ActiveWorkbook**

*Si vous oubliez cette ligne et que vous ouvrez un autre classeur par la suite, vous ne pourrez plus utiliser activeworkbook pour faire réfrence au premier classeur*

-Fermer le classeur nommé wb sans enregistrer:

Wb**.close(0)**

-Fermer le classeur nommé wb en enregistrant:

Wb**.close(1)**

-Faire un « enregistrer sous » du classeur nommé wb dans C:\Users\desk\Documents au nom classeur 1.xlsx  :

Wb**.saveas filename :=** “ C:\Users\desk\Documents\ classeur1.xlsx »

### Manipulation des Feuilles (Worksheet):

**Généralités sur les feuilles**

-Compter le nombre de feuille contenu dans un classeur:

wb.**Sheets.Count**

-Ajouter une feuille après la feuille nommé « ma\_feuille » :

wb.**Worksheets.Add After:=**worksheets(“ma\_feuille”)

-Une fois la feuille créée, **vous devez impérativement à la ligne suivante l’associer à une variable** Pour cela il faut utiliser la ligne suivante:

**Set** ws **= Activesheet**

*Si vous oubliez cette ligne et que vous manipulez une autre feuille par la suite, vous ne pourrez plus utiliser* **Activesheet** *pour faire référence à la nouvelle feuille créée*

-Supprimer la feuille attribuée à la variable ws :

**ws.Delete**

-Modifier le nom de la feuille attribuée à la variable ws :

**ws.name=**« nom souhaité »

-Supprimer tout le contenu et les formats de la feuille ws:

**ws.cells.clear**

-Supprimer tout le contenu de la feuille ws

ws.**cells.clearcontents**

-Définir le nombre de feuilles initiale **n** lors de la création d’un nouveau classeur :

**Application.SheetsInNewWorkbook** = n

-Cacher la feuille ws:

**ws.Visible = False**

-Parcourir toute les feuilles d’un classeur :

**For Each** ws **In ThisWorkbook.Worksheets**

*Instructions*

**Next** ws

**Les différentes manières de manipuler les feuilles :**

-Attribuer à ws la feuille active (ie la dernière feuille manipulée)

**Set** ws **= Activesheet**

**Inconvénient** : Vous ne savez pas exactement sur quelle feuille vous pointez. J’utilise cette ligne uniquement lorsque je crée une nouvelle feuille (cf section précédente) pour le reste, à éviter.

-Attribuer à ws la neme feuille

**Set** ws **= ThisWorkbook.Worksheets(n)**

Par exemple :



Set ws = ThisWorkbook.Worksheets(2)

Va pointer sur **sheet2**.

**Inconvénient** : Si l’utilisateur ajoute une feuille avant par exemple :



Ws pointera alors sur **sheet4**

-Attribuer à ws la feuille nommée « Sheet2 »

**Set** ws **= ThisWorkbook.Worksheets(“**Sheet2**”)**

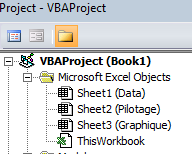
**Inconvénient** : Si l’utilisateur renomme la feuille le code plante

-La manière la plus propre est celle qui ne fait pas référence à une variable modifiable par l’utilisateur. Pour cela on va utiliser la propriété **codename** des worksheets.

Explication : Pour l’utilisateur, le nom des feuilles se résume à ce qu’il voit sur chaque onglet. Par exemple :



En ouvrant l’interface vba on peut voir :

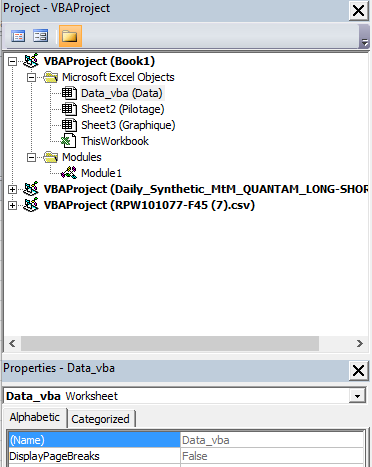


Ainsi, il existe un « nom Excel » auquel on accède grâce à ws.**name** et un « nom  VBA » auquel on accède grâce à la propriétéws.**codename**

Vous pouvez également renommer ces « noms VBA ». Par exemple pour Sheet1 :

-Cliquer sur Sheet1

-Modifier la propriété (Name) qui apparait en bas à gauche



-Attribuer à ws\_Data la feuille dont le codename est « Data\_vba »

**For Each** ws **In ThisWorkbook.Worksheets**

**If** ws.**CodeName =** "Data\_vba" **Then**

**Set** ws\_data **=** ws

**End If**

**Next** ws

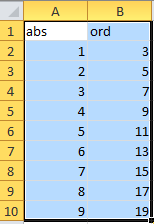
De la même manière, jusqu’à maintenant, pour faire référence à une cellule j’utilisais **thisworkbook.worksheets(1).cells(1,1**) mais là aussi on a un risque de modification du sens des feuilles par l’utilisateur. Si le nom vba de la feuille 1 est Data\_vba, il est plus robuste d’écrire **Data\_vba.cells(1,1)**

*Remarque : Excel est capricieux, parfois le fait d’utiliser directement data\_vba pour faire référence à la feuille ne fonctionnera pas pour certaines propriétés. Vous devez alors essayer les écritures précédentes. Mais utiliser au maximum cette notation.*

Exercice 8

### Manipulation des Ranges :

Un range est un ensemble de cellules par exemple la plage A1:B10 est un range. On va garder cette exemple et voir les différentes manières d’attribuer le range A1:B10 à une variable.



-Attribuer à rg la plage A1 :A10. Méthode 1 :

**Set** rg **=** thisworkbook.worksheets(1).**cells(**1**,**1**).resize(**10,2**)**

**cells(**1**,**1**)** : sera la cellule de départ (en haut à gauche du range) (ici A1)

**.resize(**10,2**)** : on étend le range de 10 lignes et de 2 colonnes

**Inconvénient** : Si l’utilisateur ajoute ou supprime des lignes, vous ne pointerez plus sur la plage désirée. Mais cette méthode reste une méthode recommandée



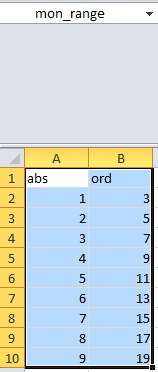
*Pour éviter ce problème on peut rendre le code plus dynamique. Par exemple, en recherchant le contenu « abs » dans toute la feuille pour connaitre la cellule de départ et compter ensuite le nombre de cellule non vide suivant abs afin de savoir de combien on étend les cellules*

-Attribuer à rg la plage A1 :A10. Méthode 2 :

Pour essayer de pallier au mieux aux interventions de l’utilisateur, vous pouvez nommer votre range en utilisant Excel.

-Selection de A1 :B10 sur la feuille

-Modification du nom en haut à gauche puis entrée

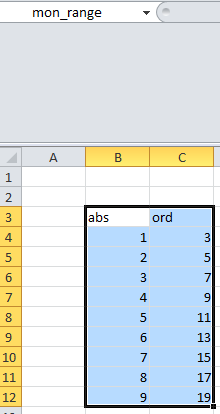


Dans mon exemple je l’ai appelé « mon\_range »

Ensuite, il suffit d’écrire la ligne suivante en vba :

**Set** rg **=** thisworkbook.worksheets(1) **.Range("**mon\_range**")**

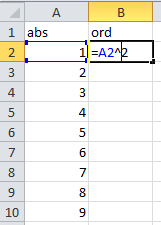
**Avantage** si on déplace votre range ou qu’on rajoute des lignes avant et/ou après, vous pointerez toujours sur le bon range



*NB : Si vous souhaitez voir les valeurs contenues dans un range à l’aide d’un espion, vous devez « espionner »* ***rg.value*** *et non pas simplement* ***rg*** *sinon vous ne verrez que les propriétés du range qui, en général, ne vous intéresse pas.*

-Etendre une formule vers le bas :

Prenons l’exemple suivant :



On a inscrit la formule B2=A2² et on souhaite étendre la formule jusqu’à B10.

Méthode :

-On set le range B2 :B10 avec les méthodes précédentes : Set rg = Sheet3.Cells(2, 2).Resize(9, 1)

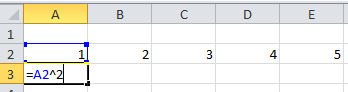
-On utilise la commande rg.FillDown

*Remarque 1 : C’est la formule contenue dans la cellule du haut qui sert de référence*

*Remarque 2 : Vous devez toujours privilégier un filldown à une boucle for car le temps d’exécution est incomparable*

*Remarque 3 : Dans l’exemple, rg ne contient qu’une colonne mais s’il contenait 50 colonnes à étendre, la ligne* rg.FillDown *aurait étendu les formules des 50 colonnes.*

-Etendre une formule vers la droite :



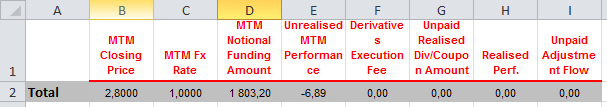
Idem :

-On set le range A3:E3 avec les méthodes précédentes : Set rg = Sheet3.Cells(3, 1).Resize(1, 5)

-On utilise la commande rg.FillRight

-Identifier une cellule par son contenu :

Prenons l’exemple suivant : Je reçois tous les jours un fichier contenant :



Je souhaite récupérer quotidiennement la valeur en référence à la colonne « MTM Notional Funding Amount », autrement dit D2.

Supposons que le classeur est déjà ouvert et que la feuille contenant les données a été attribuée à ws

**Mauvaise pratique :**

D\_result=ws.cells(2,4).value

Inconvénient : Si du jour au lendemain, quelqu’un rajoute une colonne ou une ligne de sous-totaux par exemple, votre code récupèrera une mauvaise valeur sans aucun moyen de contrôler que la valeur est fausse

**Bonne pratique :**

i\_rowresult = ws.cells(1,1).resize(100,1)**.Find(****what**:=”Total”, **lookat:=xlWhole).Row**

i\_columnresult = ws.cells(1,1).resize(1,100**).Find(what:=**” MTM Notional Funding Amount ”, **lookat:=xlWhole).column**

d\_result=ws.cells(i\_rowresult, i\_columnresult).value

Explication:

On sélectionne le range sur lequel on veut faire la recherche ws.cells(1,1).resize(100,1) on indique la valeur à chercher what:=”Total” donc ws.cells(1,1).resize(100,1).Find(what:=”Total”, lookat:=xlWhole) renvoie la cellule souhaitée et .row permet de récupérer la ligne de cette cellule.

Avantage : peu importe le numéro de colonne ou le numéro de ligne, tant qu’il y aura en colonne 1 une cellule contenant “Total, la ligne sera identifiée et tant qu’une cellule de la première ligne contiendra « **MTM Notional Funding Amount** » la bonne cellule sera récupérée.

De plus, si quelqu’un a supprimé **Total** ou **MTM Notional Funding Amount** , vous aurez un bug et vous comprendrez donc facilement d’où il vient.

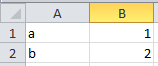
Exercice 9

## Les variables de type vba (partie 2):

Revenons sur les variables de type vba. Il existe également des variables de type tableau qui s’apparentent aux ranges des variables Excel.

Dans ce cours, on va présenter uniquement le type Variant. Ce type permet de contenir un tableau sans se soucier du type des variables contenues.

Prenons :

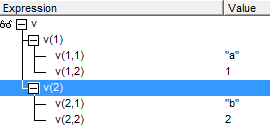


-Déclarer un variant et l’associer à un range:

Dim v As Variant

v = Sheet1.Cells(1, 1).Resize(2, 2)

On a alors :



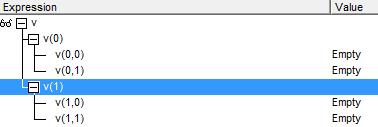
*Ici pas de set car v est une variable de type vba*

-Dimensionner un variant:

Dim v As Variant

Redim v(1,1)

On a alors :



*Attention ! Avec la première méthode l’indice commence à 1, alors qu’avec la deuxième méthode, l’indice commence à 0.*

*Pour éviter les confusions je conseille d’inscrire au début du code la ligne suivante :* **Option Base 1**

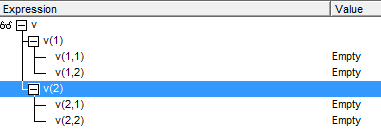
-Dimensionner un variant en base 1:

Option base 1

Dim v As Variant

Redim v(2,2)

On a alors :



Dans la suite, on suppose que la ligne option base 1 est toujours inscrite au début du code

-Remplir un Variant :

Option base 1

Dim v As Variant

Redim v(2,2)

**v(1, 1) = 2**

On a alors :



-Récupérer le nombre de lignes et de colonnes d’un variant :

Option base 1

Dim v As Variant

ReDim v(2, 3)

**i\_row = UBound(v, 1)** (=2)

**i\_column = UBound(v, 2)** (=3)

-Déclarer un tableau à une dimension (vecteur colonne) et le remplir :

**Methode 1 :**

Option base 1

Dim v As Variant

ReDim v(2, 1)

v(1, 1) = 1

v(2, 1) = 2

On a alors :



**Methode 2 :**

Option base 1

Dim v As Variant

ReDim v(2)

v(1) = 1

v(2) = 2

On a alors :



-Déclarer un tableau à 3 dimensions (cube) et le remplir :

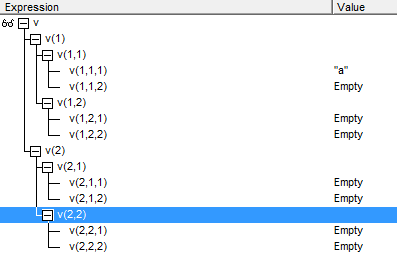
Option base 1

Dim v As Variant

ReDim v(2, 2, 2)

v(1, 1, 1) = « a »

On a alors :



-Déclarer un tableau à 2 dimensions puis le remplir et le redimensionner au fil de l’eau:

Option base 1

Dim v As Variant

ReDim v(1, 1)

v(1, 1) = 1

**ReDim Preserve v(1, 2)**

v(1, 2) = 2

Redim preserve *permet de redimensionner votre variant en conservant son contenu initial, si vous utilisez uniquement redim, alors le 1 contenu dans v(1,1) sera effacé.*

*Cela peut s’avérer utile lorsque vous ne connaissez pas à l’avance la taille finale de votre variant. Cependant il est toujours préférable en terme de temps d’exécution d’éviter de redimensionner plusieurs fois votre variant.*

***Attention ! Petite subtilité de vba, vous pouvez uniquement redimensionner la dernière dimension ie pour ReDim v(2), pour ReDim v(2, 2), pour ReDim v(2, 2, 2) etc***

Exercice 10

VBA avancé :

## Commandes supplémentaires

### Diminuer le temps d’exécution du code:

Il existe des lignes de commandes permettant de diminuer le temps d’exécution du code et/ou de limiter le risque de bug :

-Désactiver la mise à jour de l’affichage pendant que le code tourne:

Sub a()

Application.ScreenUpdating = False

…

Application.ScreenUpdating = True

End sub

*A ajouter à la version finale* *de votre projet*

-Désactiver les alertes pendant que le code tourne:

Sub a()

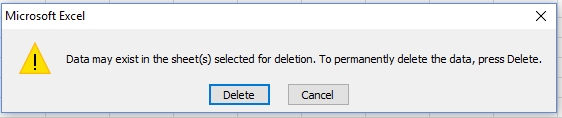
Application.DisplayAlerts = False

…

Application.DisplayAlerts = True

End sub

*Peut s’avérer utile quand vous supprimez des feuilles par exemple pour éviter l’apparition du message suivant qui bloque le déroulement du code.*



-Passer en mode calcul manuel pendant que le code tourne:

Sub a()

Application.Calculation = xlManual

…

Application.Calculation = xlAutomatic

End sub

*Si votre projet comporte beaucoup de feuille Excel avec des formules, cela permet d’éviter à Excel de refaire tous les calculs à chaque ligne de code et donc d’améliorer la rapidité d’exécution du code.*

-Mettre à jour les formules

Application.Run "RefreshAllStaticData »

OU

ThisWorkbook.RefreshAll

*Utile lorsque vous êtes en calcul manuel mais que vous souhaitez mettre à jour ponctuellement vos formules*

-Ne pas remplir le presse- papier pendant que le code tourne:

Sub a()

Application.CutCopyMode = False

…

Application.CutCopyMode = True

End sub

### Manipuler les dossiers :

-Récupérer le chemin de votre classeur :

s\_path = ThisWorkbook.Path

Renvoi :



-Créer un répertoire:

MkDir (« C:\Users\desk\Documents\**Projet\_vba »**)

Permet de créer le dossier Projet\_vba dans documents.

-Parcourir tous les fichiers excel d’un répertoire :

Rep = "C:\Documents and Settings\POSTE F\Bureau\CFAie\"

Fichier = Dir(Rep)

Do While Fichier <> ""

…

Fichier = Dir

Loop

### Gestion des erreurs :

Parfois, vous aurez des erreurs à gérer dans votre code. Par exemple, j’ouvre une centaine de fichier et je récupère certaines infos. Si un fichier est corrompu, vous ne voulez pas forcément que votre code plante mais juste qu’il ignore ce fichier.

Dans cette partie je vais présenter 3 méthodes :

**Méthode 1 :**

On Error Resume Next

Si vous inscrivez cette ligne, dans la suite du projet, si le code était censé planter à une ligne, celle-ci sera ignorée.

A éviter un maximum car souvent cela créé d’autres bugs difficiles à déceler.

**Méthode 2 :**

Cette méthode utilise la notion de balise vue précédemment.

On error goto ma\_balise

….

ma\_balise:

En cas d’erreur, le code ira directement à la ligne suivant ma\_balise :

**Méthode 3 :**

Sub main()

On error goto error

…

Exit sub

Error :

Msgbox “Suite à une erreur le programme doit s’arreter », vbcritical

End sub

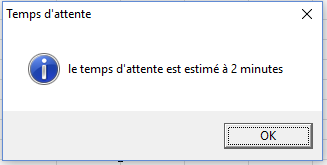
En cas d’erreur, le code ne plante pas mais sort du sub ou de la fonction

## Interactions avec l’utilisateur :

### Les boxs :

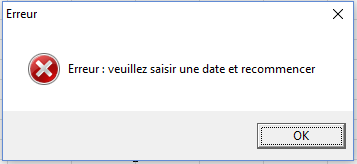
-Afficher un message d’information:

MsgBox "le temps d'attente est estimé à 2 minutes", vbInformation, "Temps d'attente"



-Afficher un message d’erreur:

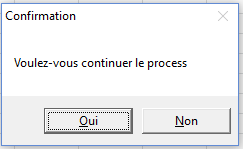
MsgBox "le temps d'attente est estimé à 2 minutes", vbcritical



*Dans les deux cas, on informe juste l’utilisateur, on ne lui laisse aucun choix*

-Poser une question dont la réponse est « oui » ou « non » à l’utilisateur:

i\_answer = MsgBox("Voulez-vous continuer le process", vbYesNo, "Confirmation")

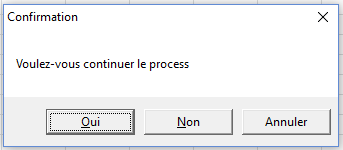


Si l’utilisateur répond « Oui » alors i\_answer=6

Si l’utilisateur répond « Non » alors i\_answer=7

- Poser une question dont la réponse est « oui » ou « non » ou « Annuler » à l’utilisateur:

i\_answer = MsgBox("Voulez-vous continuer le process", vbYesNoCancel, "Confirmation")



Si l’utilisateur répond « Oui » alors i\_answer=6

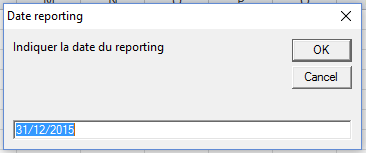
Si l’utilisateur répond « Non » alors i\_answer=7

Si l’utilisateur répond « Annuler » alors i\_answer=2

*Attention ! La syntaxe de Msgbox nécessite des parenthèses lorsqu’on stocke la réponse dans une variable. Par contre si vous ne stockez pas la réponse dans une variable (cf deux premiers exemples), vous ne pouvez pas mettre de parenthèses.*

- Poser une question dont la réponse est laissé libre à l’utilisateur:

da\_reporting = InputBox("Indiquer la date du reporting", "Date reporting", "31/12/2015")



*En général, il est préférable de mettre une valeur par défaut. Cela permet notamment de donner un « modèle » de la valeur attendue à l’utilisateur*

*Remarque : Typiquement lorsque vous utilisez des inputbox, vous allez devoir utiliser la gestion des erreurs vue précédemment. En effet, votre code n’est jamais censé planter. Si l’utilisateur saisi un nombre alors qu’une date est attendue, cette erreur doit être gérée.*

***De manière plus générale, vous devez toujours anticiper (un maximum) les modifications pouvant être faites par l’utilisateur et pouvant impacter votre macro. C’est généralement la première cause de bug***

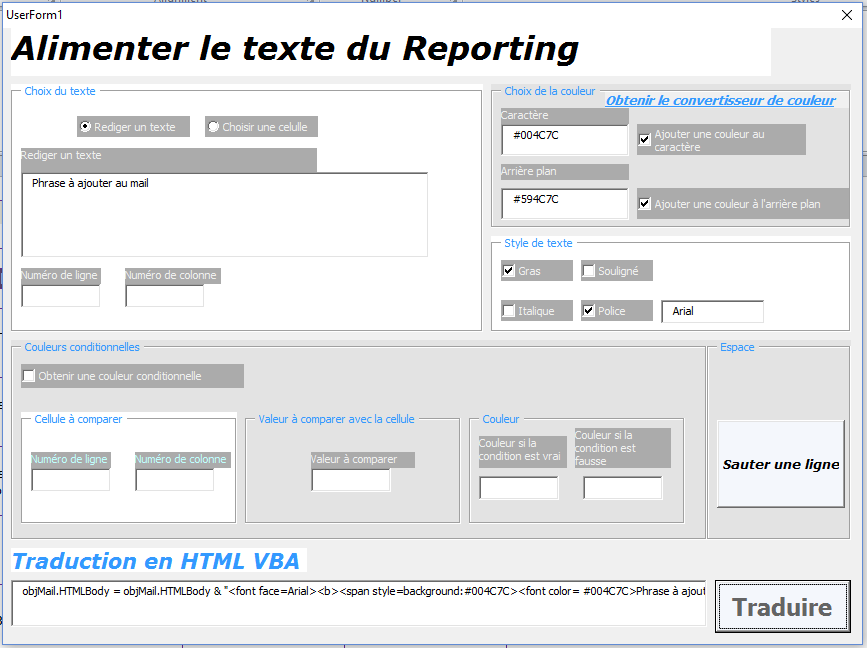
Exercice 11

### Les userforms :

Les userforms permettent une communication plus approfondie avec l’utilisateur. Ils permettent par exemple de remplir un formulaire ou ce genre de chose.

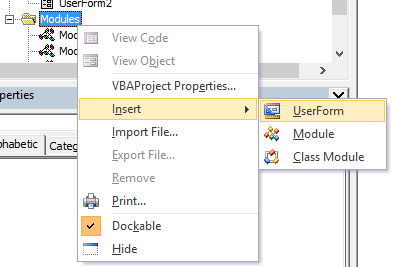
Pour tout de suite mieux comprendre l’intérêt, je vais vous présenter un travail réalisé durant mon alternance de M2 :

Le but était de générer automatiquement une ligne de code en HTML/VBA pour écrire dans un email un texte avec une mise en forme donnée

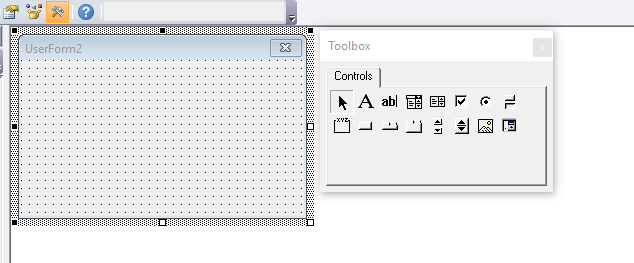


Comme vous pouvez le voir, on peut laisser de multiples choix à l’utilisateur. Nous allons voir dans la suite comment générer ce type de userform.

Commençons par insérer un userform :



La fenêtre suivante apparait :



Si la toolbox n’apparait pas, vous devez cliquer sur  .

- Redimensionner le cadre :

Il suffit de tirer sur les bords

- Ecrire un texte : 

Cliquer sur  et créer le cadre que vous voulez. Pour modifier son contenu, soit vous double cliquer sur le cadre, soit vous sélectionner le cadre et vous modifiez la propriété « Caption » en bas à gauche

- Créer une Textbox : 

Cliquer sur  et créer le cadre que vous voulez.

Une textbox servira à l’utilisateur pour inscrire un contenu libre

- Créer une ComboBox : 

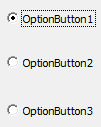
Cliquer sur  et créer le cadre que vous voulez.

Une Combobox servira à l’utilisateur pour inscrire un contenu avec une liste de choix prédéfinie (Ou un choix libre).

- Créer une CheckBox : 

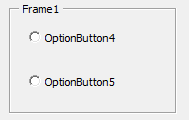
Cliquer sur  et créer le cadre que vous voulez.

- Créer une liste d’Optionbutton :



Pour créer une liste d’optionbutton vous devez d’abord créer une frame en cliquant sur .

Placer ensuite les optionbuttons dépendants les uns des autres dans la frame. Cela donne :



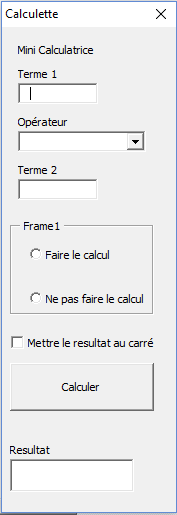
-Créer un bouton :

Cliquer sur  et créer le cadre que vous voulez.

*Remarque : Pour changer les textes, c’est toujours le paramètre Caption que vous devez modifier.*

Pour le moment, on a juste expliqué comment créer la partie graphique. Nous allons maintenant voir comment faire interagir ces figures avec le code.

On va prendre l’exemple d’une mini calculatrice de la forme suivante :



On veut :

* 2 textbox pour que l’utilisateur saisisse Terme 1 et Terme 2
* 1 Combobox contenant +,-,\*,/
* 2 optionbuttons avec « Faire le calcul » pré coché au lancement. Si l’utilisateur coche ensuite « Ne pas faire le calcul » et clique sur « Calculer », le résultat ne s’affichera pas.
* 1 Checkbox permettant de mettre le résultat de l’opération au carré si elle est cochée
* 1 Bouton « Calculer » renvoyant le résultat
* Que le userform se lance à l’ouverture du fichier Excel

-Lancer le userform à l’ouverture du fichier :

On utilise la méthode permettant de lancer une macro à l’ouverture du fichier Excel (cf Lancer une macro à l’ouverture du fichier Excel) :

Private Sub Workbook\_Open()

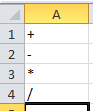
UserForm2**.Show**

End Sub

-Remplir la Combobox :

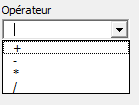
**Solution 1 : Passer par une feuille Excel**

Prendre une feuille Excel, par exemple « Sheet1 ». Mettre ce que vous voulez voir apparaitre dans la liste déroulante



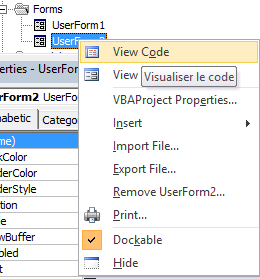
Sélectionner ensuite votre combobox et dans les propriétés, à la ligne « RowSource », inscrire le range que vous venez de remplir, ici « =Sheet1!A1:A4 ».

Lorsque vous lancez le userform, vous avez la liste remplie :



**Solution 2 : Sans passer par une feuille Excel**

-Clique droit sur le userfom -> view code



-Selectionner UserForm puis Initialize



-Le script suivant apparait



C’est dans ce sub que vous devez inscrire les lignes qui vont initialiser votre userform.

-Reste à ajouter les lignes suivantes :

Public Sub UserForm\_Initialize()

ComboBox1.AddItem "+"

ComboBox1.AddItem "\*"

ComboBox1.AddItem "-"

ComboBox1.AddItem "/"

End Sub

*Remarque : ComboBox1 correspond à la propriété (Name) et non pas à la propriété Caption. (Name) correspond au « nom vba » tandis que Caption est le contenu afficher sur le userform.*

-Pré coché l’optionbutton « faire le calcul » à l’ouverture du userform:

Il suffit d’ajouter la ligne suivante à UserForm\_Initialize :

Public Sub UserForm\_Initialize()

ComboBox1.AddItem "+"

ComboBox1.AddItem "\*"

ComboBox1.AddItem "-"

ComboBox1.AddItem "/"

**OptionButton4.Value = True**

End Sub

Passons maintenant au code qui va utiliser les différentes options déterminées par l’utilisateur pour générer le calcul

On veut que le calcul s’exécute au moment où l’utilisateur clique sur le bouton « Calculer ».

Pour cela on double clique sur « Calculer »

Le sub suivant apparait :



C’est dans ce sub que vous devez écrire votre code.

1 : récupération des valeurs terme 1 et terme 2

d\_terme1 = CDbl(TextBox1.Value)

d\_terme2 = CDbl(TextBox2.Value)

*Il ne faut pas oublier de convertir le contenu en double sinon ils seront considérés comme des strings et vous ne pourrez pas faire d’opérations dessus.*

*Ici, je simplifie le code pour l’exemple et pour uniquement me concentrer sur la manière de gérer le userform mais, en pratique, ici, on devrait avoir une gestion des erreurs prenant par exemple en compte la possibilité que l’utilisateur ait pu saisir autre chose qu’un nombre.*

2 : faire le calcul selon l’opérateur choisi dans Combobox1:

Select Case **ComboBox1.Value**

Case "+"

result = d\_terme1 + d\_terme2

Case "\*"

result = d\_terme1 \* d\_terme2

Case "-"

result = d\_terme1 - d\_terme2

Case "/"

result = d\_terme1 / d\_terme2

End Select

3 : Mettre au carré le résultat si Checkbox1 est coché :

If CheckBox1.Value = True Then result = result \* result

4 : Vérifier qu’OptionButton4 est coché pour afficher le resultat :

If OptionButton4.value = True Then TextBox3 = result

5 (Bonus) Ajouter un lien vers google en cliquant sur “Mini Calculatrice” :

Double clic sur le texte “Mini Calculatrice”

Ajouter ensuite dans le script qui apparait :

**ThisWorkbook.FollowHyperlink Address:="**http://www.google.fr"

### Les worksheetfunctions :

La grande majorité des formules d’Excel sont disponibles en VBA. En général, elles portent le même nom que les formules de la version anglaise.

Nous allons en citer quelques une :

-Fonction AVERAGE :

d\_avg = **WorksheetFunction.Average(**Feuil5.Columns(1))

-Fonction SUMPRODUCT :

d\_sumproduct = **WorksheetFunction.Average(**Feuil5.Columns(1))

-Fonction TRANSPOSE :

V\_variant = **WorksheetFunction.Transpose(mon\_range\_a\_transposer)**

**…**

Exercice 12

## Les références

Nous avons précédemment parlé des références (cf Utiliser les références).

Dans cette partie nous allons apprendre à utiliser certaines de ces références afin de manipuler le Pack Office.

### Envoyer des mails automatique : La référence Outlook

-Ajouter la référence Outlook :

Dans les références, vous devez cocher **« Microsoft Outlook 14.0 Object Library » et « Microsoft Office Outlook View Control »**

-Initialiser Outlook :

Set olApp = GetObject("", "Outlook.Application")

Dim olExplorer As Outlook.Explorer

Set objMail = olApp.CreateItem(olMailItem)

*Une fois ces lignes inscrites, vous manipulez votre email à partir de la variable objmail (que vous pouvez appeler autrement si vous le souhaitez)*

-Afficher l’email :

objMail.Display

*Si vous n’ajoutez pas cette ligne, tout se fera en masqué et le code tournera normalement plus vite*

-Ajouter l’objet du mail :

objMail.Subject = « Mon objet »

-Ajouter le corps du texte :

objMail.Body=«  corps du mail »

-Ajouter le destinataire :

objMail.To = [adresse@gmail.com](mailto:adresse@gmail.com)

-Ajouter le destinataire en copie:

objMail.CC = [adresse@gmail.com](mailto:adresse@gmail.com)

-Ajouter une pièce jointe:

objMail.Attachments.Add « C:\Users\desk\monfichier.xlsx »

-Envoyer le mail:

objMail.Send

-relâcher les variables Outlook :

Set olApp = Nothing

Set objMail = Nothing

Avec cette méthode, vous pouvez créer des mails simples. Cependant si vous souhaitez avoir une mise en forme du corps de texte, il est nécessaire de passer par du HTML.

Pour cela, vous devez :

-Ajouter avant de remplir le corps du texte : objMail.BodyFormat = olFormatRichText

-Puis remplacer objMail.Body=« corps du mail » par objMail.HTMLBody=« corps du mail »

Je vais donc donner quelques bases de HTML :

Si vous inscrivez objMail.HTMLBody ="<u><i><b>texte</b></i></u>" votre corps de mail sera : ***texte***

Explication : En HTML, on utilise des balises,<b> texte à mettre en gras </b> est la balise permettant de mettre le texte que vous placez au milieu en gras. <i> texte à mettre en italique </i> pour italique et <u> texte à souligner </u> pour souligner

Vous pouvez imbriquer autant de balises que vous le souhaitez.

-Ajouter une couleur à la police :

"<font color= #004C7C>texte à mettre en bleu</font>"

Le code couleur est #004C7C pour un bleu foncé. Pour trouver le code de la couleur que vous recherchez, vous pouvez utiliser ce lien <http://www.aliasdmc.fr/terme/exemple/convertisseur.html>

-Ajouter une couleur à l’arrière-plan:

"<span style=background: #004C7C>couleur</span>"

-Modifier la police du texte (ici Arial):

« <font face=Arial>texte en arial</font> »

-Sauter une ligne :

Pour sauter une ligne, vous devez utiliser une balise un peu particulière, il suffit d’ajouter "<br>" là où vous souhaitez sauter une ligne

### Créer des comptes rendus automatique : La référence Word

-Ajouter la référence Word :

Dans les références, vous devez cocher **« Reference : Microsoft Word 14,0 Object library »**

-Initialiser Word :

**Dim wordApp As Word.Application**

**Dim wordDoc As Word.Document**

**Set wordApp = CreateObject("word.application") 'ouvre la session word**

-Ouvrir un Document Word existant et l’associer à wordDoc :

**Set wordDoc = wordApp.Documents.Open(“**C:\Users\desk\Documents\test.docx”)

-Afficher le document word:

wordApp.Visible = True

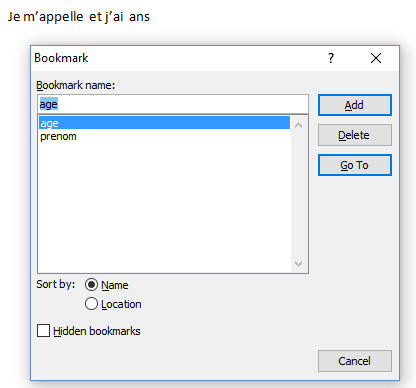
-Insérer des variables dans le texte:

Prenons l’exemple suivant, dans test.docx j’inscris le texte suivant:



J’insère ensuite un bookmark dans word (ou signet en francais) : Pour cela, je me place à l’endroit où je vais insérer une variable (ici après « je m’appelle ») puis **Insert -> Bookmark.** J’appelle mon bookmark **« prenom ».**

Je répète l’opération pour l’âge et j’appelle le signet âge.



Dans le VBA j’ajoute les lignes suivantes :

wordDoc.Bookmarks("prenom").Range.Text = "Fabien"

wordDoc.Bookmarks("age").Range.Text = "26"

Ainsi, lorsque je lance la macro, j’obtiens :



-Récupérer le nom du fichier:

S\_filename = wordDoc.Name

-fermer et sauvegarder le document:

wordDoc.Close True

-fermer le document sans sauvegarder:

wordDoc.Close False

-Enregistrer sous:

wordDoc.SaveAs Filename:=”test\_modified.docx”

-Fermer la session Word :

wordApp.Quit

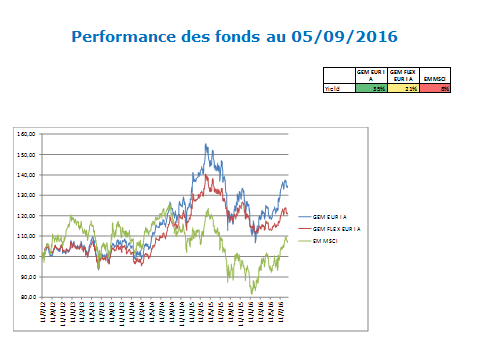
Exercice 13

### Créer des présentations automatiques : La référence Powerpoint

A partir de VBA Excel, il est possible d’automatiser à peu près tout sur Powerpoint (Création d’une présentation, ajout de slides, ajout de formes, gérer les animations etc). La difficulté est de savoir comment manipuler les objets Powerpoint car depuis la version 2007, l’enregistreur de macro sur Powerpoint a été supprimé.

Dans cette partie, je vais simplifier un maximum et vous donner quelques clés pour rendre facilement un Powerpoint dynamique. Si vous souhaitez connaitre plus de fonctionnalités je vous invite à regarder la page suivante : <http://www.globaliconnect.com/excel/index.php?option=com_content&view=article&id=168:automate-microsoft-powerpoint-from-excel-using-vba-run-a-slide-show&catid=79&Itemid=475>

Pour l’exemple, on va créer et rendre dynamique le slide suivant :



-Ajouter la référence Powerpoint :

Dans les références, vous devez cocher **« Reference : Microsoft Powerpoint 14,0 Object library »**

-Initialiser Powerpoint :

**Set** applPP **= New PowerPoint.Application**

-Afficher le document Powerpoint:

applPP**.Visible = True**

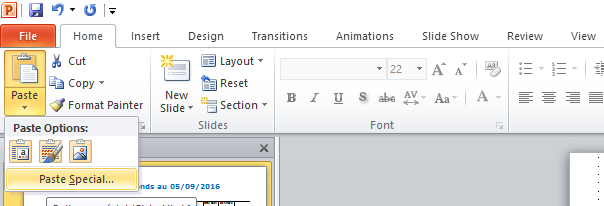
-Ouvrir un Document Powerpoint existant et l’associer à PptDoc:

**Set** PptDoc **=** applPP**.Presentations.Open("**C:\Users\desk\exemple.pptx**")**

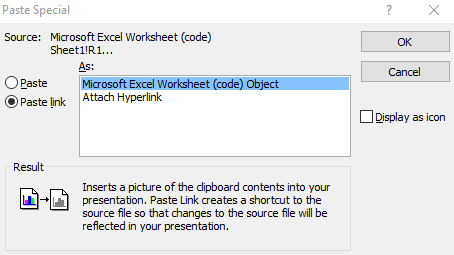
-Rendre un tableau ou un graphique dynamique :

Dans le fichier Excel contenant la macro, on crée un tableau (ou un graphique) avec les formules souhaitées. Il suffit ensuite de coller le tableau dans la présentation **avec liaison**

Pour cela, cliquer sur la flèche en haut à gauche puis Paste Special

****

Puis choisir « Paste link » et « Microsoft Excel Worksheet (code) Object » :



L’objet Powerpoint est alors relié au fichier Excel. En effet, si vous cliquez sur le tableau (ou le graphique) puis clique droit « Update links », si le graphique a changé sur la feuille Excel, il va changer sur le Powerpoint.

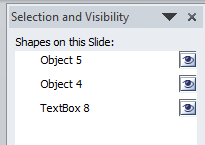
Ainsi, il vous suffit de rajouter la ligne suivante dans le code :

PptDoc.**UpdateLinks**

-Rendre une zone de texte dynamique :

Vous devez d’abord créer la zone de texte ou la forme que vous souhaitez.

En tapant ALT+F10 dans Powerpoint, la fenêtre suivante apparait à droite :



Vous pouvez visualiser les noms des objets que vous avez créés.

Il suffit alors d’ajouter la ligne suivante pour modifier le contenu de Textbox 8 (mon titre) par exemple :

PptDoc**.Slides(**1**).Shapes("**TextBox 8**").TextFrame.TextRange.Text =** "Performance des fonds au " & Date

-Enregistrer une présentation :

PptDoc**.Save**

-« Enregistrer sous » une présentation :

PptDoc**.SaveAs Filename:=** "C:\Users\desk\exemple.pptx"

Exercice 14