Visual Basic for Applications

EISTI – Ing 2 IFI

Olivier Favre

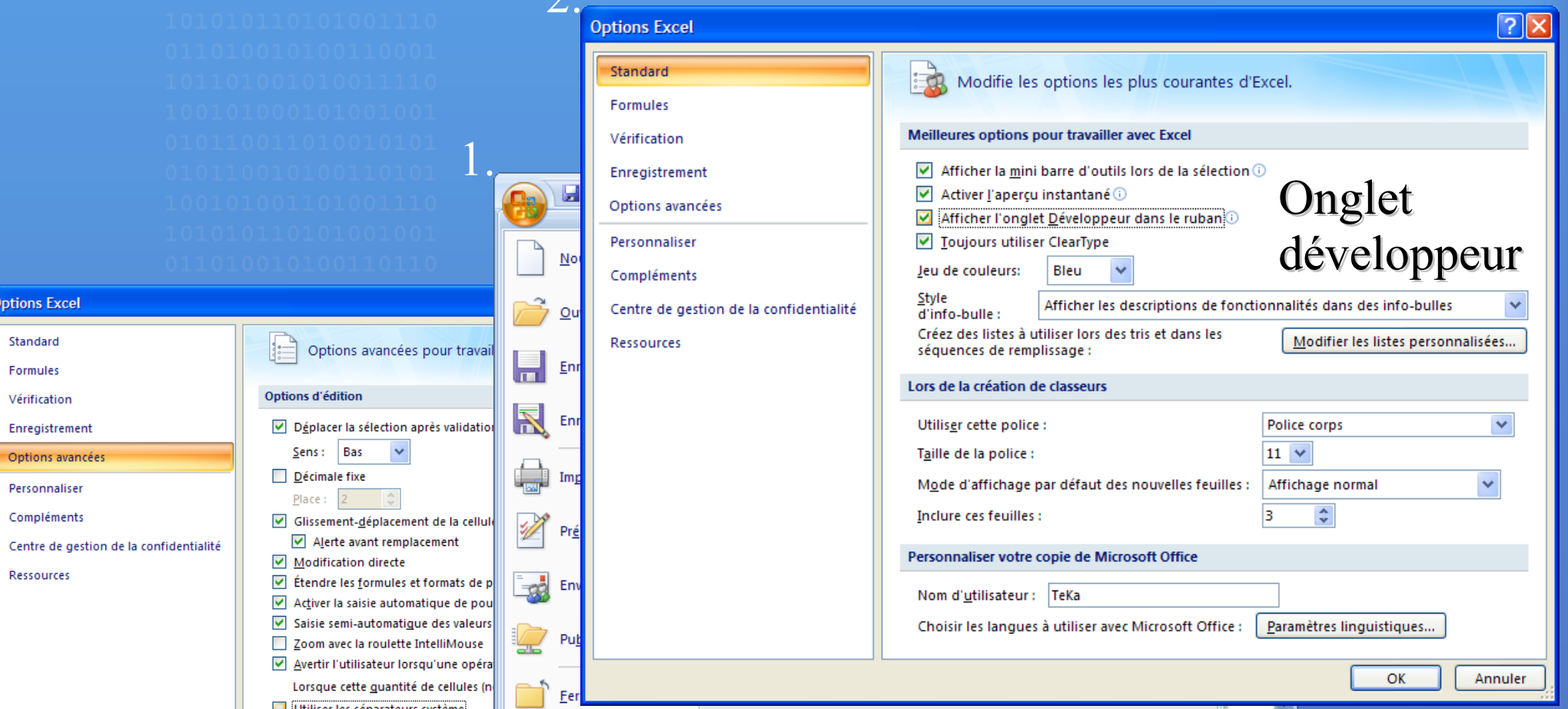
Charles Villette

Sommaire

I. Pré-requis et Formules Excel
II. Coder dans Excel : VBE, où placer le code
III. Différences VBA/VB.Net
IV. La classe Range + exo
V. La classe Application
VI. Les autres classes
VII. IHM + exo
VIII. Accès aux données + exo
IX. Code avancé : optimisations, importation de DLLs et AddIns

Pré-requis

2.



Onglet développeur

1.

3.

Séparateur à l'anglaise, Facultatif

Faisable en programmation

Aide à la conversion chaîne → nombre

Formules Excel

- Nommage absolu : `A1`
- Nommage relatif : `R1C1`, `R[-2]C`
- Nommage relatif localisé : `L1C1` (Français)
- Séparation des paramètres : « ; »
- Formules qui renvoient des matrices : Ctrl+Maj+Entrée, formule entre {} *rajoutées*
- Il existe même les macros en Français, mais c'est vraiment trop gore :-)

Visual Basic Editor

- Avant 2003, Visual Studio était séparé pour les langages. VB avait Visual Basic Editor
- VBA est directement VB 6 (version de 1998)
- Dans Office 2010... toujours 22 ans de retard
- Présentation de VBE
  - Onglet Développeur / Visual Basic (Alt+F11)
  - Explorateur d'objet (F2)
  - Complétion Ctrl+Space et Ctrl+I
  - Panneau de Projets à gauche

Où écrire le code ?

- Où le fait Excel ? → Enregistreur de macro
- Dans une feuille ?
  - Code spécifique et très local
- Dans un classeur ?
  - Événements généraux
  - Permet de créer des classeurs d'AddIns
- Dans un module ?
  - Permet l'utilisation d'une `Function` en tant que fonction dans une formule

Différences avec VB.Net et Rappels

- Propriétés avec des paramètres : `Item`
- Parenthésage : `MaSub Params 'Sans ()`
-
```
With unObjet
   .Propriété = valeur
   If .UneFonction = 0 Then 'du vrai code
End With
```
- `If UneString Like "*patt?rn" Then '...`
- `Procédure NomParam := valeur`

Différences avec VB.Net
Let, Set et DefaultProperty

- Il existe des propriétés par défaut.
  Ex : Range avec Value

```
maVar = Range("A1") 'équivalent à
maVar = Range("A1").Value
```

  UneCollection et Item()

- Récupérer un objet :

```
Dim r As Range
Set r = Range("A1")
r = Range("A1") '.Value ! → Erreur
```

Différences avec VB.Net
Les Collections

- On peut utiliser n'importe quel type en indice
- `Worksheets.Item(1) 'à .Count`
- `Worksheets(1) 'prop. par défaut`
- `Worksheets("Feuil1")`
- `Charts(Array("Chart1","Chart3"))`

Différences avec VB.Net et Spécificités VBA

- Utiliser les évennements d'un objet spécial
```
Public WithEvents Appl As Application
```
- ```
Evaluate("1+1")
Evaluate("A1")
Evaluate("SIN(A3)")
```
- ```
Debug.Print "Trace à la sortie"
Debug.Assert expressionQuiStoppeSiFaux
```
- ```
TypeName(variable)
```
- ```
Private Type MASTRUCTURE
  lMaVar As Long
  sMaVar As String
End Type
```

La classe Range
Utilisations pratiques

- `Range` : utilise l'objet actif
- `Worksheets("Feuil1").Range` : utilise la Feuil1
- Affecter plusieurs valeur **identiques** d'un coup :
```
Range("A1:D1").Value = "Toto"
```
- Affecter plusieurs valeur **différentes** d'un coup :
```
Range("A1:D1").Value = Array("Name", "Full Name", "Title", "Installed")
```

La classe Range
Principaux membres

- Activate, Select, Address, Cells, Count
- Areas > 1 en sélection multiple (indép. Forme)
- Copy/Cut [Destination], Delete, Clear, Sort
- End, Find, Range, Columns, Rows, Fill
- Value, Text, Value2, Formula
- Borders, Interior, Style, NumberFormat, *Alignment
- Parse, TextToColumns, RemoveDuplicates

Exo 1 – Première macro

- Mettre les mains dans le cambouis
- Lancer la macro :
  - Tester différents événements
  - En formule (utilise un module)
  - Appel manuel
- On teste quelques propriétés de `Range`
  - `Value, Formula, ...`

La classe Application

- La classe la plus fournie, très générale
- RegisterXLL, AddIns : Add ; Installed = True
- Calculation*, DisplayAlerts, Interactive, Visible
- Evaluate, Goto, Range, Run, Selection
- OnTime, OnKey
- Sheets, ThisCell, ThisWorkbook, Windows, Workbooks, Worksheets

Les autres classes obligatoires

- Workbooks
  - Add, Open* ; Active*, ExportAsFixedFormat (PDF), Save
- Worksheets
  - Add ; Copy, Paste, Range, Select
- Windows
  - *SideBySide* ; RangeSelection

Et le reste ?

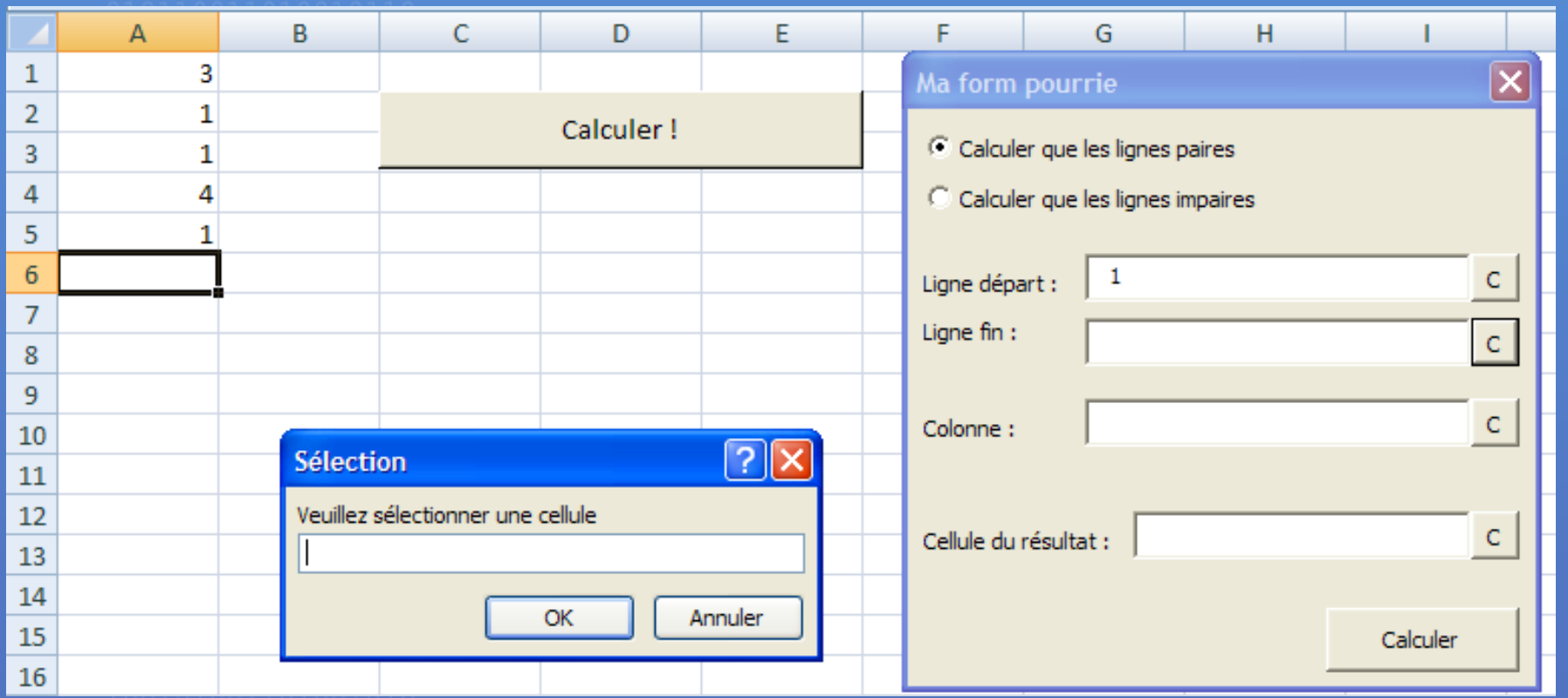
- Dans la MSDN !
- De nombreuses macros pré-existantes sont disponibles sur Internet.
- L'expérience
- Sans oublier : **La pratique !**

IHM

- Dans la feuille de calcul
- Dans une `UserForm`
- `FileDialog().Filter, ..., .Show, .SelectedItems`
- Utiliser une boîte de dialogue prédéfinie : `Application.Dialogs(xlDialogOpen).Show`
- `MsgBox, InputBox, Application.InputBox` (permet de spécifier le type des données)
- `Get{Open|SaveAs}Filename, FindFile`

Exo 2 – IHM Basique

- Objectif :
  - Calculer une somme des lignes paires/impaires
  - Créer une `UserForm` permettant à l'utilisateur de saisir des valeurs dans la feuille
  - Traiter les composants de la `UserForm` pour obtenir les informations saisies



Accès aux Bases de Données (1)

- Les base de données en excel sont très variées, il est possible d'utiliser à peu près tout et n'importe quoi, MySQL, Oracle, XML...
- Excel à plus une optique de présentation des résultats, la plupart des outils ont donc une fonction basique : récupérer toutes les entrées et les afficher dans une feuille excel, en vue de graphiques ou autres.

Accès aux Bases de Données (2)

- Façon simplifiée à l'extrême :
  - Via `OpenDatabase` :

```
Dim tmp As Workbook
Set tmp = Workbooks.OpenDatabase _
(bdd, requete)
```

    - Ceci va automatiquement ajouter les entrées trouvées dans la feuille Excel, à partir de la case A1 (début de la feuille)
    - La requête est bien entendu celle de son choix.
    - Nous sommes ici dans une importation de données proche de l'onglet données.

Accès aux Bases de Données (3)

- Via ADODb :
  - ADODb est un outils de connexion générique à une base de données (comme OleDb).
  - OLEDb et ODBC sont aussi supportés dans Excel, cependant ils n'ont nullement besoin de programmation pour être utilisés sous excel (onglet « Données »)
- Prérequis :
  - Importation des références (Outils → Référence):
    - « Microsoft Active X Data Objects 2.8... »
    - « Microsoft Active X Data Objects Recordset 2.8... »

Accès aux Bases de Données (4)

- Exemple de connexion :

```
Dim oCn As ADODB.Connection
Dim oRS As ADODB.Recordset
Dim ConnString As String,SQL As String
Dim qt As QueryTable
ConnString = _
  "Provider=Microsoft.Jet.OLEDB.4.0;" _
  & "DataSource=C:\access\mabdd.mdb"
```

Accès aux Bases de Données (5)

- Ouverture de la connexion :

```
Set oCn = New ADODB.Connection
oCn.ConnectionString = ConnString
oCn.Open
```

- Requête et envoi :

```
SQL = "SELECT * FROM client"
Set oRS = New ADODB.Recordset
oRS.Source = SQL
oRS.ActiveConnection = oCn
oRS.Open
```

Accès aux Bases de Données (6)

- Affichage dans Excel (à partir de B1):

```
Set qt = Worksheets(1) _
    .QueryTables.Add( _
    Connection := oRS, _
    Destination := Range("B1"))
qt.Refresh
```

- Fermeture & co :

```
If oRS.State <> adStateClosed Then
  oRS.Close
End If
If Not oRS Is Nothing Then Set oRS = _
                            Nothing
If Not oCn Is Nothing Then Set oCn = _
                            Nothing
```

Accès aux Bases de Données (7)

- Recordset, QueryTable et ListObject
- Recordset, vu à travers ADODb et OLEDb
- QueryTable, déjà vu
  - Tableau depuis une source de données
- ListObject
  - Correspond exactement à un « Tableau »
  - Passage de QueryTable à ListObject
  - Passage inverse ?
  - Permet ajout, suppression de lignes/colonnes, tri, calculs de total/moyenne/...

Accès aux Bases de Données (8)
Exo 3

- Appel d'une Base de Données dans VBA

Code avancé
Optimisations

- `Auto_{Exec|New|Open|Close|Exit}` (dans un module)
- `Option Explicit` : force la déclaration des variables
- `Option Base {0|1}` : indice de début des tableaux
- Désactiver MàJ écran :
  `Application.ScreenUpdating = False`
  À remettre à True à la fin de la macro !
  Inutile dans les `Function` par nature
- Désactiver Calcul Auto : `Application.Calculation = xlCalculationManual`
  Remettre à l'ancienne valeur à la fin.
- Utiliser `With`

Code avancé
Importation de DLLs

```
' Importation de la fonction de la DLL, ne peut pas être
en Public
Private Declare Function getUserName Lib "advapi32.dll" _
Alias "GetUserNameA" (ByVal lpBuffer As String, nSize _
As Long) As Long
' Macro qui l'utilise
Public Sub exempleDll()
    Dim buffer As String * 250 ' déclaration spéciale
                               'pour utiliser un vrai buffer
    getUserName buffer, 250    ' appel
    Dim usernameBuff As String ' mais les manipulations
                              'sont plus faciles avec une String
                              'classique à taille variable
    usernameBuff = buffer
    usernameBuff = Left(usernameBuff, InStr(1, _
       usernameBuff, Chr(0)) - 1) 'pour couper la String
                                  'correctement
    MsgBox """" & usernameBuff & """"
End Sub
```