Module E - Excel: VBA

Préparé par Luc Baron
Édition du 18 mars 2024

1 Introduction

Le langage VBA (Visual Basic for Application) est un langage de programmation permettant d'interagir avec les applications de la suite Office de Microsoft. Ce langage permet de programmer des tâches se déroulant à l'intérieur des applications Office. Ce document s'adresse aux lecteurs ayant déjà utilisés des feuilles de calcul Excel et ayant des bases en programmation procédurale. L'enregistrement de macros permet d'explorer la syntaxe associée à certaines commandes. Un résumé de la structure du langage VBA, ses structures de décision et de répétition, les boîtes de dialogue, les boutons de contrôle et les objets **`Workbooks`**, **`Worksheets`** et **`Range`**.

2 Macro VBA

Dans Excel, l'enregistrement d'une séquence de commandes clavier-souris dans un script VBA s'appelle une macro VBA. Plutôt que de répéter cette séquence un grand nombre fois, il est plus facile de réexécuter la séquence enregistrée. Dans cet exemple, les chiffres des ventes arrivent par courriel dans un fichier texte de format **.csv**. Le contenu de ce fichier doit être transféré dans un fichier Excel de type **.xlsx** avec un nouveau format d'affichage pour la ligne de titre et l'ajout de la ligne **Total des ventes**. Une nouvelle macro VBA sera sauvegardée dans le fichier **VentesMacro.xlsm**, un format Excel permettant de prendre en charge les macros.

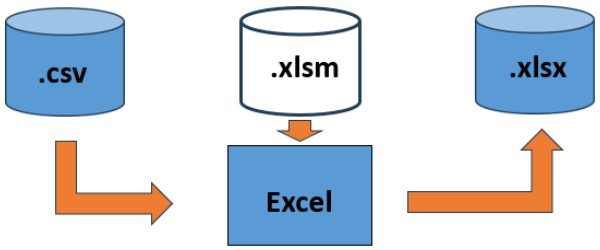
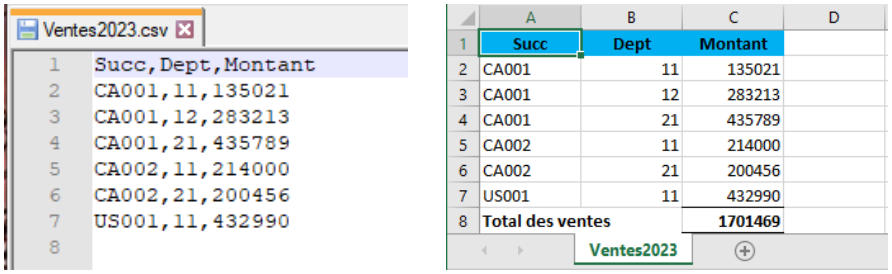


Fig. E.1 - Traitement du fichier **`Ventes2023.csv`** par la macro **`VentesMacro.xlsm`** pour produire **`Ventes2023.xlsx`**

2.1 Enregistrement

La macro VBA est enregistrée dans un nouveau classeur Excel nommé **VentesMacro.xlsm**. L'onglet *Développeur* est ouvert afin de démarrer l'enregistrement de la macro appelée **FormatVentes** avec le raccourci Ctrl-Shift-V, puis démarrer l'enregistrement. Comme montré à la Fig. E.2, les opérations suivantes sont enregistrées :

1. sélectionner **Fichier > Ouvrir**, choisir le fichier **`Ventes2023.csv`**, puis faire **OK**;
2. sélectionner **A1:C1**, puis les outils caractère gras, remplissage bleu pâle et centrer le texte;
3. sélectionner **A8**, l'outil caractère gras, inscrire **Total des ventes**, puis **[Enter]**;
4. sélectionner **C8**, l'outil caractère gras, l'outil de bordure supérieure simple et inférieure double, l'outil *Somme automatique*, puis **[Enter]**;
5. sélectionner **A1**, puis **Fichier > Enregistrer sous**, conserver le parcours et le nom du fichier, choisir le type **.xlsx**, puis **OK**;
6. sélectionner la fenêtre du classeur **VentesMacro.xlsm**, puis fin de l'enregistrement.



Ventes2023.csv

```
1  Succ,Dept,Montant
2  CA001,11,135021
3  CA001,12,283213
4  CA001,21,435789
5  CA002,11,214000
6  CA002,21,200456
7  US001,11,432990
8
```

	A	B	C	D
1	Succ	Dept	Montant	
2	CA001	11	135021	
3	CA001	12	283213	
4	CA001	21	435789	
5	CA002	11	214000	
6	CA002	21	200456	
7	US001	11	432990	
8	Total des ventes		1701469	

Ventes2023

Fig. E.2 - Ouverture et formattage du fichier **`Ventes2023.csv`**

Le classeur ouvert pendant l'enregistrement contient les données formattées et sauvegardées en format Excel **Ventes2023.xlsx**. La macro est enregistrée dans un **Module** du classeur **VentesMacro.xlsm** qui doit être sauvegardé.

2.2 Édition

Le script de la macro est accessible par **Développeur > Visual Basic**, puis ouvrir le + de **Modules** et double-clic sur **Module1** (ou autres si vous avez fait plusieurs essais). Lors de l'enregistrement d'une macro, Excel tente de convertir en instructions VBA toutes vos actions au clavier et avec la souris. Ainsi, si vous ajusté la grandeur de la fenêtre de votre classeur, faites dérouler votre feuille de calcul, ou simplement sélectionner une autre fenêtre Windows, toutes ces actions seront traduites en instructions dans votre macro, tels que `Application.Width`, `ActiveWindow.SmallScroll Down:=3` ou `Windows("Test.xlsm").Activate`. Évidemment, ces actions sont inutiles du point de vue du traitement du fichier **.csv**, et donc, ne doivent pas d'être conservées dans votre macro.

De plus lors de l'enregistrement, Excel traduit les modifications des propriétés d'un objet, mais aussi le maintient des autres propriétés à leur état actuel. Ces autres propriétés sont souvent assignées à `0`, False ou `xlNone`, et sont également inutile du point de vue du traitement désiré. Il est cependant difficile de bien distinguer entre les instructions essentielles et celles superflux. On recommande alors de procéder méthodiquement en placant graduellement en commentaire les instructions que l'on croit superflux et de vérifier que le script produit encore les bonnes opérations. Malgrés son grand nombre d'instructions en commentaires, le script ci-dessous est encore fonctionnel.

Pour la ligne de titre **A1:C1**, seul les propriétés `Font.Bold`, `Interior.Pattern`, `Interior.Color` et `HorizontalAlignment` sont nécessaires . Pour le texte **Total des ventes** en **A8**, seul la propriété `Font.Bold`, est nécessaire. Pour le total des ventes en **C8**, seul les propriétés `Font.Bold`, puis `LineStyle` et `Weight` des bordures `xlEdgeTop` et `xlEdgeBottom` sont nécessaires. Finalement, la sélection de la cellule **C9** est inutile, alors que la sélection de la cellule **A1** est conservée afin de sauvegarder un fichier **.xlsx** avec cette cellule sélectionnée par défaut. Noter la présence du symbole "_" à la fin de la dernière instruction du script. Elle indique que cette instruction se poursuit à la ligne suivante.

```
Sub FormatVentes()
'FormatVentes Macro
'Touche de raccourci du clavier: Ctrl-Shift-V
  Workbooks.Open Filename:="X:\A_Faire\Ventes2023.csv"
  Range("A1:C1")
  Selection.Font.Bold = True
  With Selection.Interior
     .Pattern = xlSolid
'     .PatternColorIndex = xlAutomatic
     .Color = 15773696
'     .PatternTintAndShade = 0
  End With
  With Selection
     .HorizontalAlignment = xlCenter
'     .WrapText = False
'     .Orientation = 0
'     .AddIndent = False
'     .IndentLevel = 0
'     .ShrinkToFit = False
'     .ReadingOrder = xlContext
'     .MergeCells = False
  End With
  Range("A8").Select
  Selection.Font.Bold = True
  ActiveCell.FormulaR1C1 = "Total des ventes:"
  Range("C8").Select
  Selection.Font.Bold = True
'  Selection.Borders(xlDiagonalDown).LineStyle = xlNone
'  Selection.Borders(xlDiagonalUp).LineStyle = xlNone
'  Selection.Borders(xlEdgeLeft).LineStyle = xlNone
  With Selection.Borders(xlEdgeTop)
       .LineStyle = xlContinuous
'     .ColorIndex = 0
'     .TintAndShade = 0
       .Weight = xlThin
  End With
  With Borders(xlEdgeBottom)
       .LineStyle = xlDouble
'     .ColorIndex = 0
'     .TintAndShade = 0
       .Weight = xlThick
  End With
'  Selection.Borders(xlEdgeRight).LineStyle = xlNone
'  Selection.Borders(xlInsideVertical).LineStyle = xlNone
'  Selection.Borders(xlInsideHorizontal).LineStyle = xlNone
  ActiveCell.FormulaR1C1 = "=SUM(R[-6]C:R[-1]C)"
'  Range("C9").Select
  Range("A1").Select
  ActiveWorkbooks.SaveAs Filename:="X:\A_Faire\Ventes2023.xlsx", _
     FileFormat:= xlOpenXMLWorkbook, CreateBackup:= False
End Sub     'ExempleEA.xlsm - Macro brute avec instructions superflux en commentaires
```

2.3 Optimisation

La dernière étape conciste à appliquer plusieurs recommandations sur l'optimisation du script. Par exemple, il faut modifier les propriétés d'un même objet avec une seule instruction `With...End With`. Dans le script final, les propriétés de caractères gras, d'aligment horizontal et de remplissage de **A1:C1** sont modifiées en une seule instruction `With`. Noter que pour la cellule **C8**, un premier niveau de `With` est utilisé pour désigner l'objet `Range("C8")` et un second niveau de `With` pour les objets `Borders(xlEdgeTop)` et `Borders(xlEdgeBottom)` séparément.

Notre script final est plus lisible et s'exécute de façon plus efficace. Cependant, il possède encore plusieurs limitations. Par exemple, le nombre de lignes de données du fichier initial ne peut pas changer. La 8e ligne de la feuille de calcul est toujours le **Total des ventes**. Les noms des fichiers d'entrée et sortie demeurent toujours les mêmes, soient **Ventes2023.csv** et **Ventes2023.xlsx**.

2.4 Recommandations

En général, l'édition et l'optimisation d'un script VBA obtenu par enregistrement d'une macro requière plusieurs étapes de transformation. Voici quelques recommandations :

1. enlever tous les `Select` inutiles;
2. regrouper dans une même structure `With...End With` les modifications de propriétés d'un même objet;
3. utiliser des variables, tels que `i` et `j`, avec la méthode `Cells(i,j)` plutôt que `Range(etiquette)`;
4. utiliser les formules relatives `R1C1`;
5. transférer ou copier-coller des données en une seule instruction (sans sélection);
6. utiliser une méthode robuste pour déterminer la dernière ligne/colonne de données.

```
Sub FormatVentes()
'FormatVentes Macro
'Touche de raccourci du clavier: Ctrl-Shift-V
  Workbooks.Open Filename:="X:\A_Faire\Ventes2023.csv"   'Ouverture du fichier .csv
  With Range("A1:C1")       'Ligne de titre
    .Font.Bold = True
    .Interior.Pattern = xlSolid
    .Interior.Color = 15773696
    .HorizontalAlignment = xlCenter
  End With
  With Range("A8")          'Texte ... Total des ventes
    .Font.Bold = True
    .FormulaR1C1 = "Total des ventes:"
  End With
  With Range("C8")          'Valeur ... Total des ventes
    .Font.Bold = True
    With .Borders(xlEdgeTop)       'Bordure supérieure
      .LineStyle = xlContinuous
      .Weight = xlThin
    End With
    With .Borders(xlEdgeBottom)    'Bordure inférieure
      .LineStyle = xlDouble
      .Weight = xlThick
    End With
    .FormulaR1C1 = "=SUM(R[-6]C:R[-1]C)" 'Formule =SUM(-6 lignes à -1 ligne)
  End With
  Range("A1").Select
  ActiveWorkbooks.SaveAs Filename:="X:\A_Faire\Ventes2023.xlsx", FileFormat:= _
     xlOpenXMLWorkbook, CreateBackup:= False  'Sauvegarde du fichier .xlsx
End Sub     'ExempleEA.xlsm - Macro optimisée - Traitement d'un fichier .csv
```

Pour déterminer la dernière ligne de données d'une feuille de calcul Excel, il faut choisir une colonne spécifique de recherche. Il est plus robuste de partir de la dernière ligne de la feuille de calcul et remonter jusqu'à la première cellule contenant une donnée. Ainsi, nous sommes assuré de trouver la dernière ligne de données de cette colonne, et ce, même s'il y a des cellules vides dans cette colonne. Ainsi, les instructions VBA suivantes permettent de remonter la colonne 3 jusqu'à la première cellule non-vide (soit la dernière ligne de données) ou se déplacer vers la gauche sur la ligne 4 jusqu'à la première cellule non-vide (soit la dernière colonne de données).

```
DerniereLigne = Cells(Rows.Count, 3).End(xlUp).Row           'Dernière ligne de la colonne 3
DerniereColon = Cells(4, Columns.Count).End(xlToLeft).Column 'Dernière colonne de la ligne 4
```

3 Programmation VBA

En VBA, les objets sont organisés dans une hiérarchie qui reflète la structure des applications Office. Voici une vue générale de la hiérarchie des objets VBA dans Excel :

1. **Application** : Au sommet de la hiérarchie se trouve l'objet **Application**. Il représente l'application Office spécifique (par exemple, Excel, Word, Access) dans laquelle le code VBA s'exécute. Vous pouvez accéder à cet objet en utilisant **Application** ou directement par son nom, comme **Excel.Application** ;
2. **Workbook** (Classeur) : Dans Excel, un **Workbook** représente un classeur Excel. Vous pouvez avoir plusieurs classeurs ouverts dans une instance d'Excel. Vous accédez aux classeurs à l'aide de la collection **Workbooks** ;
3. **Worksheet** (Feuille de calcul) : À l'intérieur d'un classeur Excel, vous avez des **Worksheets** (feuilles de calcul) qui représentent les onglets. Vous pouvez accéder à ces feuilles de calcul à l'aide de la collection **Worksheets** ou **Sheets** du classeur ;
4. **Range** (Plage) : Un **Range** représente une plage de cellules dans une feuille de calcul. Vous pouvez accéder aux cellules, les lire et les modifier à l'aide d'un objet **Range** ;
5. **Chart** (Graphique) : Si vous travaillez avec des graphiques dans Excel, vous pouvez accéder à ces objets via la collection **Charts** d'une feuille de calcul.

Cette hiérarchie des objets permet de naviguer à travers les différentes parties d'une application et de manipuler les données et les éléments d'interface utilisateur de manière programmée. Chaque niveau de la hiérarchie a ses propres *propriétés*, *méthodes* et *événements* que vous pouvez utiliser pour automatiser des tâches et personnaliser votre application.

3.1 Structure du langage

3.1.1 Types des variables

Par défaut, il n'est pas nécessaire de déclarer les variables avant leurs utilisations. Elles sont toutes initialement de type Variant, c'est-à-dire à ajustement automatique de leur type selon les données que leurs assignes. Cependant, il est une bonne pratique de forcer la déclaration des variables avec l'instruction `Option Explicit` au début de vos procédures VBA.

Dans tous les cas, les noms de variable doivent commencer par une lettre, ne pas contenir les caractères spéciaux ! . @ & $ #, ni utiliser un mot réservé du langage VBA, tel que `Dim`, `As`, `Byte`, `If`, `Then`, `Else`, `End If`, `Do`, `Loop`, `For`, etc, et qui sont généralement affichés en bleu dans l'éditeur VBA. Les variables de types numériques peuvent être déclarées de la façon suivante :

```
Dim a As Byte     'Entier de 0 à 255
Dim b As Integer  'Entier de -32768 à 32767
Dim c As Long     'Entier entre +/- 2.15 milliard
Dim d As Single   'Nombre à point flottant simple précision
Dim e As Double   'Nombre à point flottant double précision
Dim f As Currency 'Nombre à point fixe à 4 décimales entre +/- 9.2 trillion
```

Les autres types de variables peuvent être déclarées de la façon suivante :

```
Dim g As Boolean  'Logique vrai ou faux (True/False)
Dim h As String   'Chaîne de 1 à 65400 caractères
Dim i As Date     'Valeur de date entre 1 jan. 100 à 31 déc. 9999
Dim j As Variant  'Type à ajustement automatique (par défaut)
Dim k As Object   'Tous les objets Excel (Workbook, Worksheet, Range, Chart, etc)
```

3.1.2 Opérateur arithmétique

Il existe 5 opérateurs arithmétiques, c'est-à-dire :

```
A + B  'Addition de A et B
A - B  'Soustraction de A par B
A * B  'Multiplication de A et B
A / B  'Division de A par B
A ^ B  'Exponentiation de A par B (A exposant B)
```

3.1.3 Expression booléenne

Les expressions booléennes ne peuvent prendre que la valeur de **Vrai** ou **Faux**. Elles utilisent des opérateurs de comparaison et des opérateurs logiques, afin d'exprimer les critères de décision. Ainsi, il existe 6 opérateurs de comparaison, c'est-à-dire :

```
  10 >  8     'Vrai, puisque 10 est plus grand que 8
  10 <> 8     'Vrai, puisque 10 est différent de 8
  10 <= 8     'Faux, puisque 10 n'est pas plus petit ou égale à 8
  10 >= 10    'Vrai, puisque 10 est plus grand ou égale à 10, ici égale
 "A" <  "B"   'Vrai, puisque A est plus petit que B en ordre alphabétique
"MEC" = "GCH" 'Faux, puisque MEC n'est pas identique à GCH
```

3.1.4 Opérateur logique

Il existe 4 opérateurs logiques, soient : et, ou, ou exclusif et négation, c'est-à-dire :

```
A And B  'Vrai, si A et B sont tous les deux vrai
C Or D   'Vrai, si au moins C ou D est vrai (vrai si les 2 sont vrai)
C Xor D  'Vrai, si seulement C ou D est vrai (faux si les 2 sont vrai)
Not E    'Vrai, si E est faux
```

3.1.5 Fenêtre de débogage

La **fenêtre immédiate** de l'éditeur VBA est un outil essentiel pour le débogage de script VBA. Pour cette raison, elle est souvent appelée *fenêtre de débogage*. Cette fenêtre est rendue accessible via l'éditeur VBA en choisissant: **Affichage > Fenêtre** immédiate ou par le raccourci *Ctrl-G* (**Windows**) et *Ctrl-Cmd-G* (**MacOS**). Heureusement, la fenêtre immédiate et l'éditeur VBA ne sont pas affichés lors de l'exécution normale d'un script VBA, ainsi l'usager n'a pas connaissance des messages imprimés dans celle-ci.

Avec la fenêtre immédiate, il est possible d'interagir avec VBA en demandant l'exécution d'instructions spécifiques, de procédures ou de fonctions. Il est possible d'interroger VBA sur la valeur de constantes ou de variables. Il est également possible d'inclure dans votre script l'instruction `Debug.Print` afin d'imprimer dans cette fenêtre le suivi de l'avancement de l'exécution. Finalement, il est possible d'interroger le contenu des variables suite à l'insertion d'un point d'arrêt (Break point). Dans ce document la fenêtre immédiate est représentée en jaune, les commandes usagers en vert et les réponses de VBA en noir. Il exist également plusieurs autres outils de débogage dans l'éditeur VBA, et qui sont pas décrit dans ce résumé.

Dans la fenêtre immédiate ci-dessous, la commande `Print 11 <= 12` demande d'évaluer l'expression logique `11 <= 12` et d'afficher le résultat, c'est-à-dire `True`. La commande `r = MaLogique(False)` demande l'exécution de la fonction `MaLogique()` avec `False` en entrée et retourne le résultat dans la variable `r`. La commande immédiate `? r` demande d'afficher le contenu de la variable `r`. Le script de la fonction `MaLogique()` reçoit et retourne des variables de type booléennes. À la ligne 2, les variables booléennes `b` et `c` sont créés pour la durée de l'exécution de la fonction. À la ligne 3, `b` est évalué à faux, puisque `b = Not (4 <> 5) = Not (True) = False`. À la ligne 4, l'instruction `Debug.Print` permet d'afficher dans la fenêtre de débogage le contenu des variables `a` et `b`. À la ligne 5, `c` est évalué à faux, puisque `c = a Or b = False Or False = False`. À la ligne 6, `MaLogique` est évaluée à vrai, puisque `MaLogique = "a" < "b" Xor c = True Xor False = True`. À la ligne 7, l'instruction `Debug.Print` permet d'afficher dans la fenêtre de débogage le contenu des variables `c` et `MaLogique`.

```
Function MaLogique(a As Boolean) As Boolean
  Dim b, c As Boolean
  b = Not 4 <> 5
  Debug.Print "[1] a=" & a & " b=" & b
  c = a Or b
  MaLogique = "a" < "b" Xor c
  Debug.Print "[2] c=" & c & " MaLogique=" & MaLogique
End Function   'ExempleEB.xlsm - Fonction qui reçoit et retourne un vrai ou faux
```

```
Print 11 <= 12
True
r = MaLogique(False)
[1] a=False b=False
[2] c=False MaLogique=True
? r
True
```

3.1.6 Valeur et format des cellules

La cellule est l'unité fondamentale des feuilles de calcul Excel. Chaque cellule peut contenir rien, une valeur ou une formule et possède un format d'affichage, tel que la fonte, le remplissage, les bordures ou la grandeur. En VBA, l'objet `Range` permet de désigner une ou plusieurs cellules. La méthode `Cells` de l'objet `Range` permet d'indexer les cellules avec un numéro de ligne et de colonne `Range.Cells(2,3)` plutôt qu'avec une étiquette `Range("C2")`. Puisque que `Cells` est la méthode par défaut de l'objet `Range`, ce dernier peut être omis, et donc, `Cells(2,3)` est suffisant.

Le contenu des cellules peut être manipulés avec la propriété `Value`. Par exemple, l'instruction `Range("A1").Value = Cells(1,2).Value` permet de lire le contenu de la cellule **B1** (ligne 1, colonne 2) et d'inscrire cette valeur à la cellule **A1**. Ce transfert de contenu se fait à chaque fois que l'instruction VBA est exécutée. Si le contenu de **B1** change sans que l'instruction soit exécutée à nouveau, alors la cellule **A1** ne contiendra pas la même valeur que la cellule **B1**. Puisque que `Value` est la propriété par défaut de l'objet `Range` (et aussi `Cells`), alors cette propriété peut être omise, et donc, `Range("A1") = Cells(1,2)` est suffisant.

Les 6 instructions suivantes permettent toutes d'assigner la valeur 8 à la cellule **C2**. À la ligne 1, les objets `Workbooks("Classeur1.xlsx")` et `Worksheets("Feuil1")` sont deux objets parents de `Range("C2")`. Ils peuvent être omis, s'ils sont tous les deux actifs. À la ligne 2, la propriété `Value` n'est pas nécessaire puisqu'elle est la propriété par défaut de `Range`. À la ligne 3, on assume que le nom **MaCellule** fait référence à **C2**. Aux lignes 4 et 5, avec ou sans l'objet `Range`, la méthode `Cells` est utilisée pour indexer **C2**. Finalement, la méthode `Offset(1,2)` produit un décalage relativement à un objet de référence. Par exemple, `Cells(2,3)` est obtenue par décalage de 1 ligne et 2 colonnes relativement à `Cells(1,1)`, c'est-à-dire `Cells(1,1).Offset(1,2)`. Il suffit d'additionner les indices.

```
Workbooks("Classeur1.xlsx").Worksheets("Feuil1").Range("C2").Value = 8
Range("C2") = 8             'La propriété Value est optionnelle
Range(MaCellule) = 8        'Le nom MaCellule fait référence à C2
Range.Cells(2,3) = 8        'La méthode Cells permet d'utiliser des numéros de ligne et colonne
Cells(2,3) = 8              'La collection Cells peut être utilisée directement sans l'objet Range
Cells(1,1).Offset(1,2) = 8 'Le décalage demande l'addition des lignes et colonnes
```

Les 6 instructions suivantes permettent toutes d'assigner la valeur 7 à la plage **C2:E3**. À la ligne 1, la valeur 7 est assigné à toutes les cellules de la plage **C2:E3**. À la ligne 2, la propriété **`Offset`** est utilisée pour décaler la plage de référence **A1:C2** de 1 ligne vers le bas et 2 colonnes vers la droite. À la ligne 3, le nom **MaPlage** fait référence à **C2:E3**. À la ligne 4, l'objet **`Range`** utilise les cellules du coin supérieur gauche **`Cells(2,3)`** et inférieur droit **`Cells(3,5)`** afin de désigner **C2:E3**. À la ligne 5, la plage **C2:E3** est obtenu par la méthode **`Resize(1,2)`**, c'est-à-dire l'agrandissement à 1 ligne et 2 colonnes à partir de **`Cells(2,3)`** au coin supérieur gauche. Finalement à la ligne 6, la même plage est obtenue par décalage de **A1** de 1 ligne vers le bas et 2 colonnes vers la droite pour obtenir **C2**, puis l'agrandissement de 1 ligne et 2 colonnes.

```
Range("C2:E3") = 7                      'C2:E3 = 7 (2 lig., 3 col.)
Range("A1:C2").Offset(1,2) = 7          'C2:E3 = (A1:C2) décalé de 1 lig., 2 col. = 7
Range(MaPlage) = 7                      'Le nom MaPlage fait référence à C2:E3
Range(Cells(2,3),Cells(3,5)) = 7        'C2:E3 = Cells(2,3) à Cells(3,5) = 7
Cells(2,3).Resize(1,2) = 7              'C2:E3 = C2 ajoute 1 lig., 2 colonnes = 7
Cells(1,1).Offset(1,2).Resize(1,2) = 7 '(C2 = A1 décalé de 1 lig., 2 col.) ajoute 1 lig., 2 col.
```

Les instructions suivantes permettent de modifier les propriétés **`Font.Bold`**; **`Font.Italic`**; **`Font.Color`**, et la propriété de remplissage **`Interior.Color`** de **`Range`** et **`Cells`**. La ligne 1 active les caractères gras de la fonte sur **B2:B6**. La ligne 2 désactive les caractères italic de la fonte de **B2:B6**. La ligne 3 assigne la couleur rouge à la fonte de **B2:B6**. Les 3 premières lignes peuvent être avantageusement remplacées par l'instruction **`With`**...**`End With`** qui permet d'éviter de répéter l'objet **`Range("B2:B6")`** et la propriété **`Font`** sur lequel plusieurs modifications de propriétés sont faites. La dernière ligne assigne le brun **`RGB(213,153,49)`** comme couleur de remplissage de **A1:E2**..

```
Range("B2:B6").Font.Bold = True       'Caractère gras de B2:B6
Range("B2:B6").Font.Italic = False    'Pas de caractère italic de B2:B6
Range("B2:B6").Font.Color = vbRed     'Caractère rouge de B2:B6
With Range("B2:B6").Font              'Idem aux lignes 1 à 3 sur B2:B6
  .Bold = True
  .Italic = False
  .Color = vbRed
End With
Cells(1,1).Resize(1,4).Interior.Color = RGB(213,153,49)  'Remplissage en brun de A1:E2
```

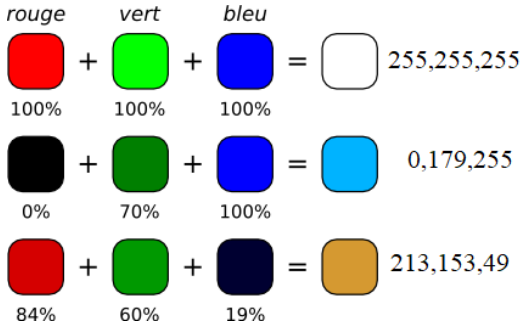


Fig. E.3 - Mélange des intensités de couleurs rouge, vert et bleu (RGB)

Tel que montré à la Fig. E.3, la fonction **`RGB(red,green,blue)`** permet de spécifier toutes les couleurs par un mélange d'intensité de rouge (0-255), vert (0-255) et bleu (0-255). Par exemple, le rouge est obtenu par **`RGB(255,0,0)`** et le brun par **`RGB(213,153,49)`**. Les couleurs standards sont définies par :

`vbBlack` : noir	**`vbWhite`** : blanc	**`vbRed`** : rouge	**`vbGreen`** : vert
`vbBlue` : bleu	**`vbYellow`** : jaune	**`vbCyan`** : cyan	**`vbMagenta`** : magenta

3.2 Structure de décision

En VBA, il existe deux structures de décision permettant de choisir le groupe d'instructions à exécuter. La structure **`If...End If`** est utile lorsque le nombre de choix est petit. La structure **`Select Case...End Select`** est utile lorsque le nombre de choix est plus grand. La décision est prise par l'évaluation d'une expression booléenne.

3.2.1 Décision If...End If

La structure de décision **`If...End If`** (ci-dessous) permet d'évaluer une séquence d'expressions booléennes afin de déterminer lequel des groupes d'instructions sera exécuté. Un seul groupe d'instruction sera exécuté. Ainsi, si l'expression booléenne 1 est vrai,

alors l'instruction 1 est exécutée, puis on saute après le `End If`. Sinon l'expression booléenne 2 est évaluée. Si elle est vrai, l'instruction 2 est exécutée, puis on saute après le `End If`. Sinon, l'instruction 3 est exécutée. L'instruction 3 est exécutée si et seulement si les deux expressions booléennes 1 et 2 sont fausses.

```
If expression_booléenne1 Then
  Instruction1
ElseIf expression_booléenne2 Then
  Instruction2
Else
  Instruction3
End If
```

La procédure **CelluleNonVide("B1")** affiche dans la fenêtre de débogage le message *Cellule B1 est non-vide*, si effectivement la cellule **B1** contient quelque chose. La procédure **AffichePositif(3,4)** permet de vérifier si le contenu de la cellule **D3** (ligne 3 et colonne 4) est supérieur à zéro et d'inscrire le mot "Positif" à la cellule de droite (c'est-à-dire **E3**). Sinon on inscrit le mot "Non-positif" dans **E3**. La procédure **ValeurDiscriminant(3.2)** permet d'afficher dans la fenêtre de débogage un des 3 messages selon le signe du discriminant.

```
Sub CelluleNonVide(Cellule AS String)  'Usage: CelluleNonVide("B1")
  If Not IsEmpty(Range(Cellule).Value) Then
    Debug.Print "Cellule " & Cellule & " est non-vide"
  End If
End Sub           'ExempleEB.xlsm - Message de cellule non vide
```

```
Sub AffichePositif(Ligne, Colonne AS Integer)  'Usage: Call AffichePositif(3,5)
  If Cells(Ligne,Colonne) > 0 Then
    Cells(Ligne,Colonne + 1) = "Positif"
  Else
    Cells(Ligne,Colonne + 1) = "Non positif"
  End If
End Sub           'ExempleEB.xlsm - Affiche si positif ou non positif
```

```
Sub ValeurDiscriminant(Discriminant AS Double)  'Usage: ValeurDiscriminant(3.2)
  If Discriminant > 0 Then
    Debug.Print "Discriminant positif."
  ElseIf Discriminant = 0 Then
    Debug.Print "Discriminant zéro."
  Else
    Debug.Print "Discriminant négatif."
  End If
End Sub           'ExempleEB.xlsm - Affiche le signe du discriminant
```

3.2.2 Décision Select Case...End Select

La structure de décision `Select Case...End Select` (ci-dessous) permet d'évaluer une variable selon plusieurs critères de décision afin de choisir le groupe d'instructions à exécuter. Un seul groupe d'instructions sera exécuté. Ainsi, si la variable est égale au critère 1, alors l'instruction 1 est exécutée, puis on saute après le `End Select`. Sinon, le critère 2 est évalué. S'il est vrai, alors l'instruction 2 est exécutée, puis on saute après le `End Select`. Finalement, si aucun des critères n'est respecté, alors l'instruction 3 est exécutée.

```
Select Case Variable
  Case Critère1
  Instruction1
  Case Critère2
  Instruction2
  Case Else
  Instruction3
End Select
```

3.3 Boîtes de dialogues

En VBA, il existe plusieurs boîtes de dialogues, dont 3 sont fréquemment utilisées: 1) l'entrée d'une valeur ou d'un texte dans une boîte de dialogue; 2) l'affichage d'un message dans une boîte de dialogue; et 3) la sélection d'un fichier dans un répertoire. Ces opérations se font avec les objets ou méthode **`InputBox`**, **`MsgBox`** et **`FileDialog`**.

3.3.1 Questionner l'usager (`InputBox`)

L'objet **`InputBox`** permet : 1) d'ouvrir une boîte de dialogue avec un titre; 2) d'afficher une question avec un espace réponse; et possiblement 3) une réponse par défaut déjà inscrite dans l'espace réponse, tel que montré à la Fig. E.4 de gauche. L'objet **`InputBox`** retourne la chaîne de caractères saisie. L'instruction `Debug.Print` permet d'imprimer de l'information dans la fenêtre immédiate de débogage de l'éditeur VBA. En usage normal, cette fenêtre n'apparaît pas, et donc, cette information de débogage n'est pas montrée à l'usager. Le symbole **`&`** permets d'assembler (concaténer) des chaînes de caractères.

```
Dim Saisie As String
Saisie = InputBox("Question ? ", "Titre", "Réponse_par_défaut")
Debug.Print "Vous avez saisi : " & Saisie & "."
```

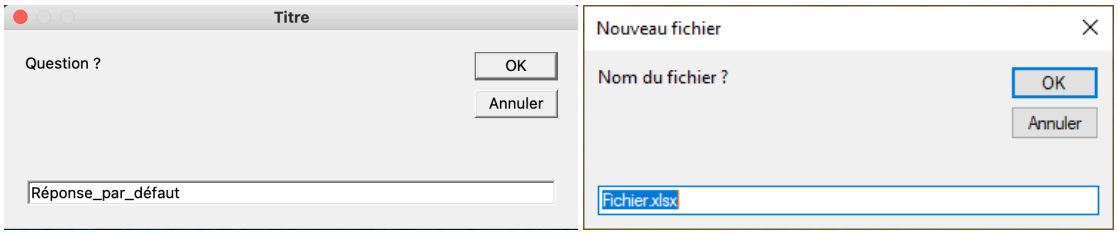


Fig. E.4 - Boîte de dialogue **InputBox** sous **MacOS** et **Windows**

Comme montré à la Fig. E.4 de droite, la fonction **LireNomFichier()** permet d'afficher la boîte de dialogue de titre *Nouveau fichier* contenant la question *Nom du fichier ?* avec par défaut le nom *Fichier.xlsx*. Un clic sur **OK** ou **Enter** accepte le nom par défaut ou le nouveau nom entré. Le message *Le nom du fichier saisi ...* est alors affiché dans la fenêtre immédiate de débogage. Un clic sur **Annuler** permet de quitter la boîte de dialogue avec une saisie vide. Le message *Opération annulée ou saisie vide* est alors affiché dans la fenêtre immédiate de débogage.

```
Function LireNomFichier() AS String   'Usage: Nom = LireNomFichier()
  Dim NomFichier As String
  NomFichier = InputBox("Nom du fichier ? ", "Nouveau fichier", "Fichier.xlsx")
  If NomFichier = "" Then          'Vérifie si la saisie est vide
    Debug.Print "Opération annulée ou aucune saisie."
  Else
    Debug.Print "Le nom du fichier saisi est : " & NomFichier & "."
  End If
  LireNomFichier = NomFichier      'Retourne le nom saisi
End Sub            'ExempleEB.xlsm - Utilisation de InputBox()
```

```
Nom = LireNomFichier()
Le nom du fichier saisi est : Fichier.xlsx
```

3.3.2 Informer l'usager (MsgBox)

L'objet **MsgBox** permet : 1) d'ouvrir une boîte de dialogue avec un titre; 2) d'afficher un message, 3) d'afficher des boutons de contrôles, et possiblement, une icône, tel que montré à la Fig. E.6. La variable **vbBoutons** de type **vbMsgBoxStyle** défini les boutons et l'icône à utiliser. La variable **vbReponse** de type **vbMsgBoxResult** récupère le bouton sélectionné.

```
Dim vbBoutons As vbMsgBoxStyle
Dim vbReponse As vbMsgBoxResult
vbBoutons = ...
vbreponse = MsgBox("Message", vbBoutons, "Titre")
If vbReponse = ... Then
  actions ...
End If
```

La variable **vbBoutons** de type **vbMsgBoxStyle** peut prendre les valeurs :

- **vbOKOnly** (0): Affiche le bouton **OK** seulement;
- **vbOKCancel** (1): Affiche les boutons **OK** et **Annuler**;
- **vbAbortRetryIgnore** (2): Affiche les boutons **Abandonner**, **Recommencer**, **Ignorer**;
- **vbYesNoCancel** (3): Affiche les boutons **Oui**, **Non** et **Annuler**;
- **vbYesNo** (4): Affiche les boutons **Oui** et **Non**;
- **vbRetryCancel** (5): Affiche les boutons **Recommencer** et **Annuler**;

et aussi ajouter les valeurs d'icônes suivantes :

- **vbCritical** (16): Affiche une icône d'erreur (X cercle rouge);
- **vbQuestion** (32): Affiche une icône de question (? cercle bleu);
- **vbExclamation** (48): Affiche une icône d'exclamation (! triangle Jaune);
- **vbInformation** (64): Affiche une icône d'information (i cercle bleu)

La variable **vbReponse** de type **vbMsgBoxResult** peut prendre les valeurs :

- **vbOK** (1): Si le bouton **OK** est sélectionné;
- **vbCancel** (2): Si le bouton **Annuler** est sélectionné;
- **vbAbort** (3): Si le bouton **Abandonner** est sélectionné;
- **vbRetry** (4): Si le bouton **Recommencer** est sélectionné;
- **vbIgnore** (5): Si le bouton **Ignorer** est sélectionné;
- **vbYes** (6): Si le bouton **Oui** est sélectionné;
- **vbNo** (7): Si le bouton **Non** est sélectionné.

Comme montré à la Fig. E.5, le script VBA ci-dessous affiche 4 fenêtres **`MsgBox`** de style différentes. La première ligne affiche la fenêtre *Bonjour* avec un *Titre*. La seconde affiche la fenêtre *Appuyer sur Enter pour continuer* avec *Pause temporaire* en titre et le bouton **OK**. La 3e ligne définit la variable **`vbReponse`** de type **`vbMsgBoxResult`**. La 4e ligne affiche la fenêtre *Voulez-vous supprimer le fichier ?* avec *Confirmation* en titre, les boutons **Oui** et **Non** et le triangle jaune (icône **`vbExclamation`**). Finalement, la 5e ligne définit la variable **`vbBoutons`** de type **`vbMsgBoxStyle`**, afin de spécifier (à la ligne 6) les boutons **`vbRetryCancel`** et l'icône **`vbQuestion`**. La ligne 7 affiche la fenêtre *Fichier introuvable. Réessayer ?* avec *Validation* en titre. La ligne 8 vérifie si **`vbReponse`** est le bouton **`vbRetry`** et affiche dans la fenêtre de débogage le message *Recommancer la recherche*, sinon *Abandonner la recherche*. Noter que les icônes sont affichées uniquement sur **Windows**, aucune icône n'est affichée sur **MacOS**.

```
MsgBox "Bonjour",, "Titre"
MsgBox "Appuyer sur Enter pour continuer", vbOKOnly, "Pause temporaire"
Dim vbReponse As vbMsgBoxResult
vbReponse = MsgBox("Voulez-vous supprimer le fichier ?", vbYesNo + vbExclamation, "Confirmation")
Dim vbBoutons As vbMsgBoxStyle
vbBoutons = vbRetryCancel + vbQuestion
vbReponse = MsgBox("Fichier introuvable. Réessayer ?", vbBoutons, "Validation")
If vbReponse = vbRetry Then
  Debug.print "Recommancer la recherche"
Else
  Debug.Print "Abandonner la recherche"
End If            'ExempleEB.xlsm - Utilisation de MsgBox()
```

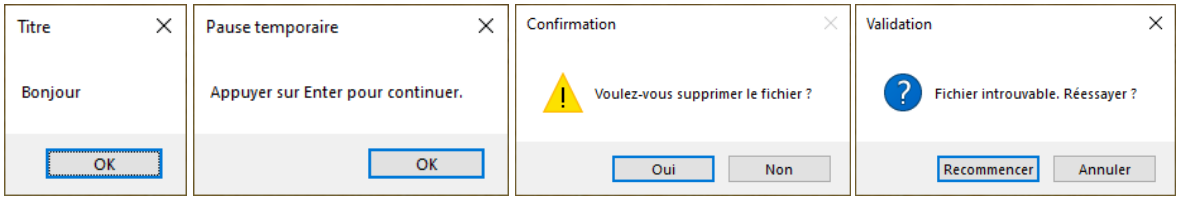


Fig. E.5 - Boîtes de dialogue **`MsgBox`** selon 4 styles différents sous **Windows**

3.3.3 Choisir un fichier ou un répertoire (`FileDialog`)

La méthode **`FileDialog`** de l'objet **`Application`** permet : 1) d'ouvrir une boîte de dialogue; 2) de naviguer à travers les répertoires; et 3) de choisir un fichier ou un répertoire, tel que montré à la Fig. E.6. L'objet **`MaBoite`** de type **`FileDialog`** permet de contenir une nouvelle instance de choix d'un fichier (**`msoFileDialogFilePicker`**) ou de choix d'un dossier (**`msoFileDialogFolderPicker`**). L'instruction **`Debug.Print`** permet d'imprimer le nom sélectionné que l'on récupère par **`MaBoite.SelectedItems(1)`**.

```
Dim MaBoite As FileDialog
Set MaBoite = Application.FileDialog(msoFileDialogFilePicker)  'Nouvelle instance de FileDialog
Debug.Print "Vous avez choisi " & MaBoite.SelectedItems(1)     'Affiche le nom sélectionné
```

La procédure **NomFichier()** affiche une boîte de dialogue permettant de parcourir et choisir un fichier. Le nom du fichier sélectionné (*X:\A_Faire\Classeur1.xlsx*) est imprimé dans la fenêtre de débogage. Cette méthode n'est fonctionnelle que sur **Windows**.

```
Sub NomFichier()              'Usage: Call NomFichier()
  Dim MaBoite As FileDialog
  Dim CheminFichier As string
  'Assigne à MaBoite une nouvelle instance de FileDialog
  Set MaBoite = Application.FileDialog(msoFileDialogFilePicker)  'Recherche un fichier
  If MaBoite.Show = -1 Then                    'Est-ce qu'un nom a été choisi
    CheminFichier = MaBoite.SelectedItems(1)  'Récupère le nom choisi
    Debug.Print "Vous avez sélectionné : " & CheminFichier
  Else
    Debug.Print "Opération annulée par l'usager"
  End If
End Sub           'ExempleEB.xlsm - Utilisation de FileDialog()
```

```
Call NomFichier()
Vous avez sélectionné : X:\A_Faire\Classeur1.xlsx
```

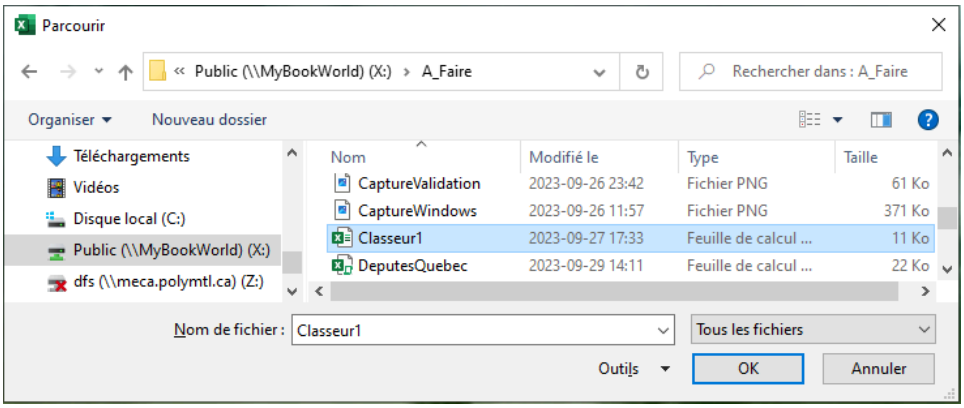


Fig. E.6 - Boîte de dialogue **`FileDialog`** sous **Windows** seulement

3.4 Structure de répétition

En VBA, il existe six structures permettant de répéter l'exécution d'instructions selon l'information disponible. Les deux structures **`For...Next`** sont utiles lorsque le nombre de répétitions est connu. Les quatre structures **`Do...Loop`** sont utiles lorsque le nombre de répétitions est variable.

3.4.1 Répétition For Each...In...Next

La structure de répétition **`For Each...In...Next`** (ci-dessous) permet de parcourir les éléments d'une liste afin d'effectuer une opération sur ceux-ci. Les éléments peuvent être les cellules d'une feuille, les feuilles d'un classeur ou même les classeurs d'un répertoire. Pour chaque élément de liste, les instructions 1 à 2 sont exécutées. Durant le traitement de chaque élément, si la condition de sortie immédiate devient vrai, alors l'instruction **`Exit For`** permet d'interrompre la répétition et quitter immédiatement la structure de la boucle sans exécuter l'instruction 2, ni de traiter les autres éléments.

```
For Each element In Liste
  Instruction1
  Condition_de_sortie_immédiate Exit For
  Instruction2
Next element
```

Tel que montré à la Fig. E.7, la procédure **AfficheRouge("B2:B6")** affiche en rouge les cellules de la liste **B2:B6**, dont le montant est supérieur à 50. Noter la présence obligatoire de l'instruction **`Set`** qui créer l'instance **`Plage`** avec l'objet **`Range(Adresses)`**.

```
Sub AfficheRouge(Adresses As String)  'Usage: AfficheRouge("B2:B6")
  Dim Cellule, Plage As Range
  Set Plage = Range(Adresses)
  For Each Cellule In Plage
    If Cellule > 50 Then
      Cellule.Font.Color = vbRed
    End If
  Next Cellule
End Sub           'ExempleEB.xlsm - Affiche en rouge les montants > 50
```

	A	B	C	D	E
1	No Produit	Montrant			
2	S45-A	123 $			
3	P13-Z	65 $			
4	P14-Z	23 $			
5	V67-A	15 $			
6	B57-S	112 $			

Fig. E.7 - Résultats de l'exécution de la procédure **AfficheRouge("B2:B6")**

3.4.2 Répétition For...Next

La structure de répétition **`For...Next`** (ci-dessous) permet de répéter des instructions avec un compteur assigné à une valeur de début, une valeur finale, et optionnellement, un pas d'avancement. Les instructions 1 à 2 sont exécutées à répétition avec le **compteur** variant de **debut** à **fin** par incrément de **pas**. Durant les répétitions, si la condition de sortie immédiate devient vrai, alors l'instruction **`Exit For`** permet d'interrompre la répétition et quitter immédiatement la structure de la boucle sans exécuter l'instruction 2, ni d'exécuter les autres répétitions.

```
For compteur = debut To fin Step pas
  Instruction1
  Condition_de_sortie_immédiate Exit For
  Instruction2
Next compteur
```

Tel que montré à la Fig. E.8, la procédure **Diagonale(5)** inscrit dans la feuille les nombres de 1 à 5 en diagonale, soit les cellules **A1**, **B2**, **C3**, ... **E5**. La fonction **valeur = Factoriel(5)** calcul le factoriel de 5, soit 5!=120. Noter qu'il s'agit d'une nouvelle fonction Excel qui peut désormais être utilisée dans une cellule avec **=Factoriel(5)**. La procédure **FiligrammePaire(5)** applique un remplissage vert (indexe de couleur 35) aux lignes paires pour les colonnes 1 à 4. La dernière ligne de la colonne `c=5` est déterminée avec la méthode `.End(xlUp).Row` appliquée à l'objet `Cells(Rows.count,c)`. Une façon robuste de trouver la dernière ligne de données d'une colonne.

```
Sub Diagonale(m As Integer)  'Usage: Diagonale(5)
  For i = 1 To m
    Cells(i,i) = i
  Next i
End Sub            'ExempleE1B.xlsm - Affiche les nombres 1 à m en diagonale
```

```
Function Factoriel(n As Long) As Long  'Usage: valeur = Factoriel(5)
  For i = n-1 To 1 Step -1
    n = n * i
  Next i
  Factoriel = n
End Function       'ExempleE1B.xlsm - Calcul le factoriel de n
```

```
Sub FiligrammePaire(c As Integer)  'Usage: FiligrammePaire(5)
  DerniereLigne = Cells(Rows.Count, c).End(xlUp).Row 'Trouve la dernière ligne de la colonne c
  For i = 2 To DerniereLigne Step 2
    Cells(i, 1).Resize(1, 4).Interior.ColorIndex = 35 'Étendre la plage à 1 ligne par 4 colonnes
  Next i
End Sub            'ExempleEB.xlsm - Affiche les lignes paires en vert des 4 premières colonnes
```

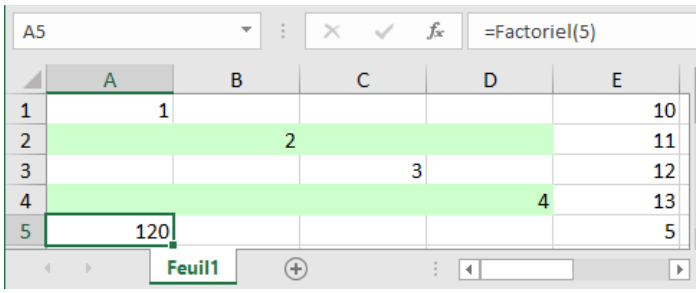


Fig. E.8 - Résultats de l'exécution des procédures et la fonction à la cellule **A5**

3.4.3 Répétition Do While...Loop

La structure de répétition `Do While...Loop` permet d'exécuter des instructions en spécifiant la condition d'entrée à la structure. Si la condition d'entrée est fausse, aucune instruction de la structure ne sera exécutée. Si la condition d'entrée est vrai, les instructions sont exécutées une 1ière fois avant de vérifier si la condition d'entrée est encore vrai avant d'exécuter à nouveau les instructions. Les répétitions se poursuivent tant et aussi longtemps que la condition d'entrée demeure vrai. Noter que les instructions de la structure peuvent ne jamais s'exécuter si la condition d'entrée est fausse. Les instructions 1 à 2 sont exécutées tant et aussi longtemps que **`condition_d'entrée`** est vrai. Lorsque celle-ci devient fausse, les répétitions cessent. Durant les répétitions, si la condition de sortie immédiate devient vrai, alors l'instruction `Exit Do` permet d'interrompre la répétition et quitter immédiatement la structure de la boucle sans exécuter l'instruction 2, ni d'exécuter les autres répétitions.

```
Do While condition_d'entrée
  Instruction1
  Condition_de_sortie_immédiate Exit Do
  Instruction2
Loop
```

La procédure **Alphabet(6)** permet d'afficher une séquence de 6 fenêtres contenant le message *Lettre: A Continuer?* avec les boutons **Oui** et **Non**. Un clic sur **Oui** permet d'afficher la fenêtre avec la lettre suivante ou **Non** pour cesser la séquence. La fonction **`chr(65)`** désigne la caractère "A". La fonction **valide = ValideMDP("MEC1315")** demander et valide que le mot de passe de l'usager est bien "MEC1315". La fonction retourne vrai si le bon mot de passe est entré, sinon elle retourne faux. Une fois le mot de passe entré, la fenêtre *Connexion à cogito validée!* s'affiche si le bon mot de passe est entré, sinon la fenêtre *Mot de passe invalide. Réessayer?* s'affiche avec les boutons **Recommencer** et **Annuler**. Un clic sur **Recommencer** permet de retourner à la fenêtre d'entrée du mot de passe et un clic sur **Annuler** permet de quitter la fonction sans la validation. La condition d'entrée de la structure est **`ValideMDP=False`** et **`a`** est différent de **`vbCancel`**, c'est-à-dire: **`a<>vbCancel`** ou de façon équivalente **`Not a=vbCancel`**. On quitte la structure lorsque **`ValideMDP`** devient vrai ou lorsque l'usager clic sur le bouton **Annuler**, c'est-à-dire: **`a=vbCancel`**.

```
Sub Alphabet(n As Integer)  'Usage: Alphabet(6)
  i = 0
  a = vbYes
  Do While i < n And a = vbYes
    a = MsgBox("Lettre: " & chr(65+i) & " Continuer ?", vbYesNo, "Alphabet des " & n & _
       " premières lettres")
    i = i + 1
  Loop
End Sub           'ExempleEB.xlsm - Affiche les n premières lettres de l'alphabet
```

```
Function ValideMDP(MotDePasse As String) As Boolean  'Usage: valide = ValideMDP("MEC1315")
  Dim mdp As String
  ValideMDP = False
  Do While Not ValideMDP And Not a = vbCancel
    mdp = InputBox("Mot de passe: ", "Serveur cogito")
    If mdp = MotDePasse Then
      ValideMDP = True
      a = MsgBox("Connexion à cogito validée!", vbOKOnly, "Serveur cogito")
    Else
      a = MsgBox("Mot de passe invalide. Réessayer?", vbRetryCancel, "Serveur cogito")
    End If
  Loop
End Function      ExempleEB.xlsm - Demande un mot de passe tant que invalide et pas Cancel
```

La structure de répétition `Do...Loop While` utilise une condition à la fin de la structure permettant l'exécution des instructions au moins une fois avant de vérifier la condition de répétition. Une condition de sortie immédiate de la structure peut être utilisée.

```
Do
  Instruction1
  Condition_de_sortie_immédiat Exit Do
  Instruction2
Loop While Condition_de_répétition
```

Tel que montré à la Fig. E.9, la procédure **Alphabetv2(6)** effectue les mêmes actions que la procédure **Alphabet(6)** avec une ligne en moins. Il n'est pas nécessaire d'initialiser `a = vbYes` afin d'assurer la condition d'entrée à la structure, puisqu'ici elle sera initialisé durant de la première répétition. Cette seconde version est une amélioration et montre l'utilité de choisir une structure avec condition d'entrée ou condition de répétition.

```
Sub Alphabetv2(n As Integer)  'Usage: Alphabetv2(6)
  i = 0
  Do
    a = MsgBox("Lettre: " & chr(65+i) & " Continuer ?", vbYesNo, "Alphabet des " & n & _
       " premières lettres")
    i = i + 1
  Loop While i < n And a = vbYes
End Sub           'ExempleEB.xlsm - Affiche les n premières lettres de l'alphabet Version 2
```

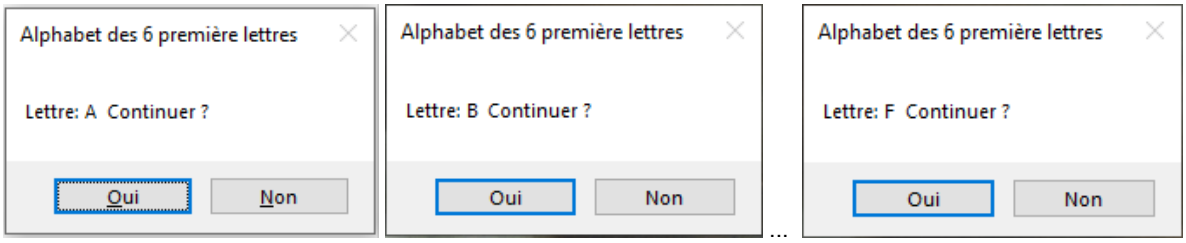


Fig. E.9 - Quelques boîtes messages de la procédure **Alphabetv2(6)**

3.4.4 Répétition Do Until...Loop

La structure de répétition `Do Until ...Loop` utilise une condition de sortie au début de la structure permettant l'exécution des instructions jusqu'à ce que la condition de sortie devienne vrai. Une condition de sortie immédiate de la structure peut être utilisée.

```
Do Until condition_de_sortie
  Instruction1
  Condition_de_sortie_immédiat Exit Do
  Instruction2
Loop
```

La structure de répétition `Do...Loop Until` utilise une condition de sortie à la fin de la structure permettant l'exécution des instructions au moins une fois et jusqu'à ce que la condition de sortie devienne vrai. Une condition de sortie immédiate de la structure peut être utilisée.

```
Do
  Instruction1
  Condition_de_sortie_immédiat Exit Do
  Instruction2
Loop Until condition_de_sortie
```

Tel que montré à la Fig. E.10, la fonction **valide = ValideMDPv2("MEC1315")** effectue les mêmes actions que la fonction **ValideMDP("MEC1315")** avec les mêmes lignes, sauf pour la condition de sortie à la fin de la structure de répétition. Le critère de sortie s'exprime de façon plus directe (sans les négations) et constitue donc une amélioration relativement à la version précédente.

```
Function ValideMDPv2(MotDePasse As String) As Boolean  'Usage: valide = ValideMDPv2("MEC1315")
  Dim mdp As String
  ValideMDPv2 = False
  Do
    mdp = InputBox("Mot de passe: ", "Serveur cogito")
    If mdp = MotDePasse Then
      ValideMDPv2 = True
      a = MsgBox("Connexion à cogito validée!", vbOKOnly, "Serveur cogito")
    Else
      a = MsgBox("Mot de passe invalide. Réessayer?", vbRetryCancel, "Serveur cogito")
    End If
  Loop Until ValideMDPv2 Or a = vbCancel
End Function      ExempleEB.xlsm - Demande un mot de passe jusqu'à validé ou Cancel
```

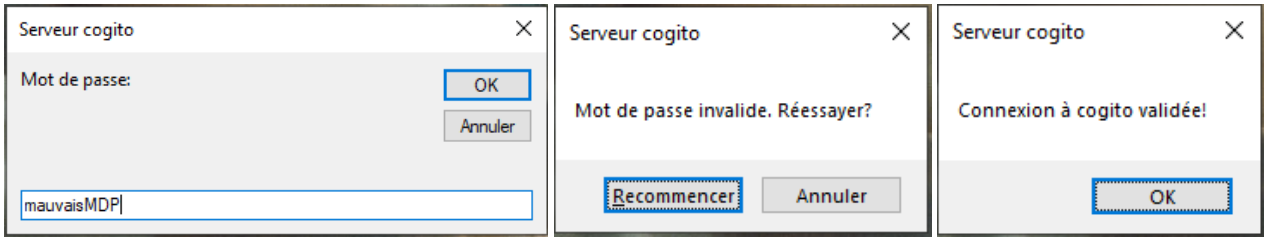


Fig. E.10 - Les 3 boîtes de dialogue de la fonction **ValideMPDv2()**

3.5 Bouton Contrôles de formulaire

Pour utiliser une feuille de calcul comme un formulaire, il est fréquent d'inclure un bouton de type **Contrôles de formulaire** afin de démarrer l'exécution d'une procédure VBA. Comme montré à la Fig. E.11, les opérations suivantes sont nécessaires afin d'insérer un bouton dans une feuille de calcul et le relier à une procédure VBA :

1. sélectionner **Insérer** dans le ruban *Développeur*;
2. sélectionner le **bouton rectangulaire** dans la catégorie **Contrôles de formulaire**;
3. sélectionner un coin, puis glisser vers le coin opposé, du bouton dans la feuille de calcul;
4. accepter la nouvelle macro VBA **Bouton1_Cliquer()** avec **Nouvelle**;
5. faire un clic-droit sur le bouton afin d'afficher le menu contextuel;
6. sélectionner **Modifier le texte**, inscrire *Bouton Bonjour*, puis cliquer ailleurs sur la feuille; et finalement
7. ajouter l'instruction `Debug.Print "Bonjour"` dans la procédure **Bouton1_Cliquer()** de l'éditeur VBA.

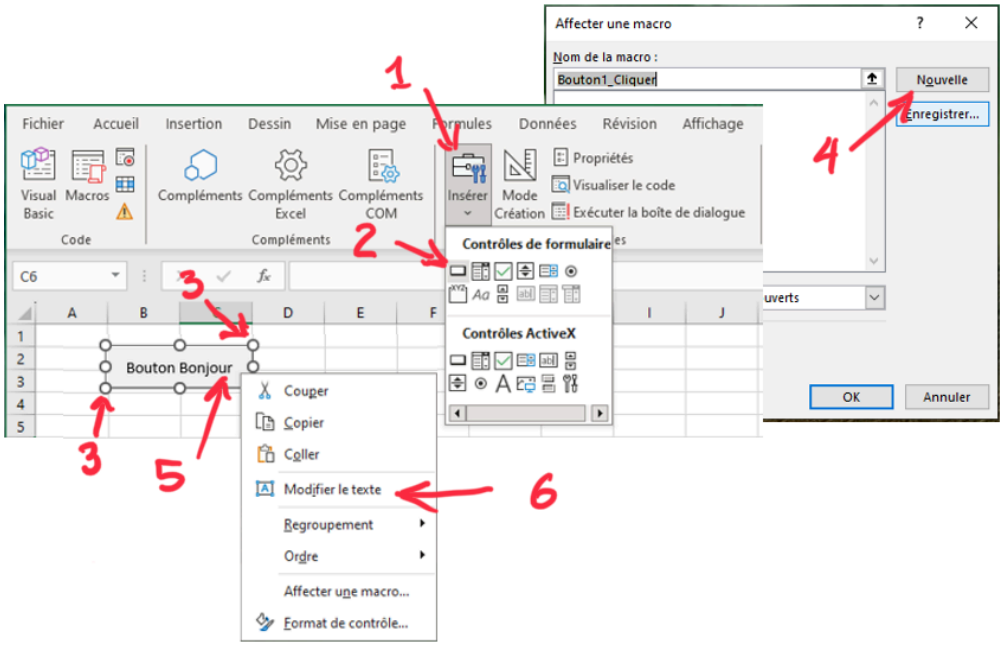


Fig. E.11 - Ajout d'un bouton de contrôles de formulaire

```
Sub Bouton1_Cliquer()
  Debug.Print "Bonjour"
End Sub   'ExempleEB.xlsm - Procédure exécutée par un clic sur un bouton dans une feuille de
calcul
```

```
Bonjour
```

Chaque clic sur le bouton *Bouton Bonjour* demande l'exécution de la procédure **Bouton1_Cliquer()** qui elle affiche le mot **Bonjour** dans la fenêtre de débogage. L'item **Format de contrôle...** du menu contextuel permet de modifier le format d'affichage du bouton (fonte, couleur, etc). L'item **Affecter une macro...** permet de relier le bouton à une autre procédure VBA. Il est également possible d'utiliser des boutons **Contrôles ActiveX** de **Windows** qui eux offrent beaucoup plus de propriétés et d'évènements (autres qu'un simple clic sur le bouton), mais qui ne fonctionnent que sur **Windows**.

4 Applications

4.1 Intégrale d'une fonction f(x)

Un script VBA doit être préparé afin de calculer l'intégrale d'une fonction f(x) entre x1= -2 et x2= +2. Ainsi, nous avons :

f(x) = 3 cos(x a) + 3 pour x < -1 et x > 1;
f(x) = 3 autrement;

avec a = Pi / 2. Tel que montré à la Fig. E.12, la colonne **A** décrit les valeurs de x de -2 à + 2 par incéremnt de 0.1, soit de la cellule **A2** à **A42**. La nouvelle fonction VBA, appelé **MaFonction()**, sera utilisée directement dans les cellules du d'Excel. Dans ce cas, la fonction ne peut pas faire référence directement ou indirectement aux autres cellules, ni de faire des interactions telles **MsgBox()** ou Debug.Print, sauf par ses paramètres d'entrées ou sorties. La nouvelle fonction doit utiliser en entrée une référence à la cellule de gauche (telle que **A2**) pour obtenir sa valeur x. Ainsi, il sera possible de copier cette fonction vers d'autres lignes sans avoir à manuellement modifier la référence, celle étant automatiquement modifée par la copie vers d'autres lignes. Dans notre cas, on utilise une référence par valeur ByVal plutôt que par référence ByRef parce que l'on désire obtenir la valeur sans modifier le contenu de la cellule d'origine. La valeur de Pi est obtenue par **`WorksheetFunction.Pi()`**.

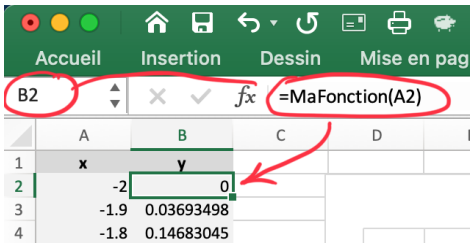


Fig. E.12 - Appel de =MaFonction(A2)

```
Function MaFonction(ByVal x As Range) As Double
  Dim a, dblPi As Double
  dblPi = WorksheetFunction.Pi()      'Valeur système pour Pi à 15 décimales
  a = dblPi / 2                       'Calcul de Pi / 2
  If x < -1 Or x > 1 Then             'Idem pour x.Value
    MaFonction = 3 * Cos( x * a) + 3 'Si x<-1 ou x>1
  Else
    MaFonction = 3                    'Si x>=-1 et x<=1
  End If
End Function   'ExempleEC.xlsm
```

Tel que montré à la Fig. E.13, la cellule **M6** contient la formule **=Trapeze(B2:B42) * h / 2**, soit le calul de l'intégrale par la méthode des trapèzes avec les valeurs de y contenues dans la plage **A2:A42**, alors que la cellule **M5** a été renommé **h**. Le script VBA ci-dessous créer une fonction VBA appelé **Trapeze()** qui sera utilisée directement dans les cellules du d'Excel. Le nombre de ligne de **`plageY`** est obtenu par la méthode **`Rows.Count`**. La boucle For est effectuée pour chacune des cellules **`y`** de la plage **`PlageY`**.

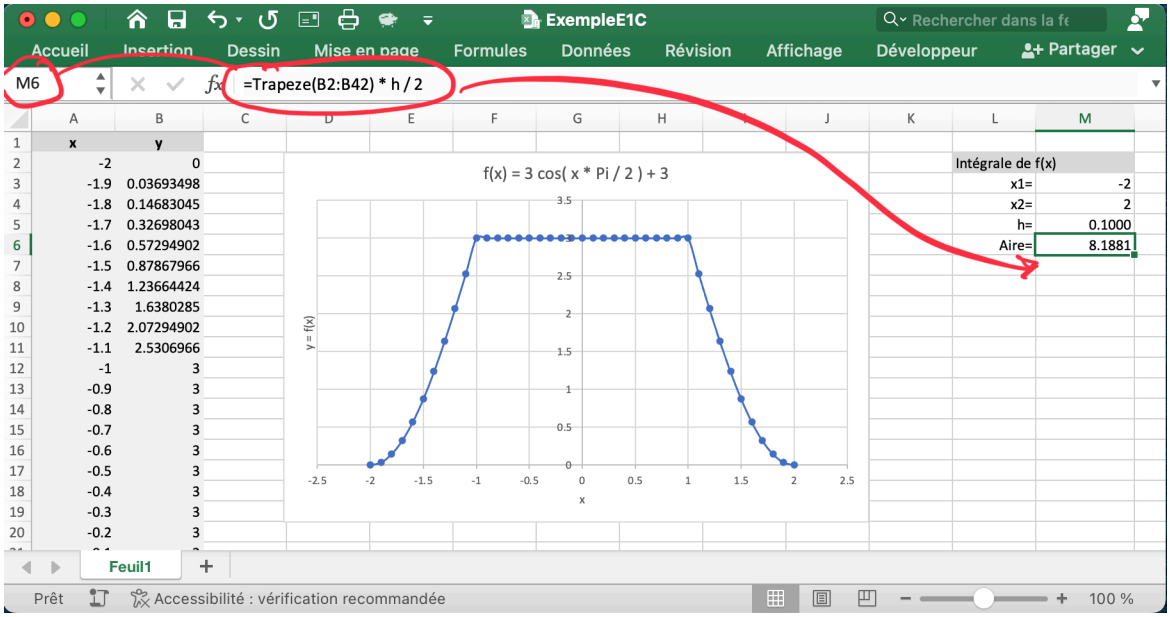


Fig. E.13 - Affichage de la fonction et calcul de l'intégrale

```
Function Trapeze(ByVal PlageY As Range) As Double
  Dim somme As Double
  n = PlageY.Rows.Count      'Compte le nombre de ligne de la plage Y
  somme = 0 : i = 0
  For Each y In PlageY       'Pour chaque cellule de la plage Y
    If i = 0 Or i = (n - 1) Then
      somme = somme + y      'Si premier ou dernier, on utilise 1 fois y (idem y.Value)
    Else
      somme = somme + 2 * y 'Sinon on utilise 2 fois y
    End If
    i = i + 1
  Next y
  Trapeze = somme
End Function    'ExempleEC.xlsm
```

4.2 Solution des moindres carrés

En VBA, les vecteurs et matrices sont traités comme des tableaux de données Excel. En général, il est préférable de les déclarer avec l'instruction Dim de type Variant sans en spécifier les dimensions, puis d'utiliser l'instruction ReDim pour ajuster la dimension de ceux-ci selon le contexte du programme. Ainsi, la solution de s moindres carrés du système algébrique linéaire surdéterminé

`M a = b`

se calcul par le pseudo-inverse gauche de la matrice **`M`** de la façon suivante

`a = inv(Mt M) Mt b`,

où **`Mt`** est la transposé de la matrice **`M`**.

La fonction **`MoindresCarres()`** permet de solutionner un système algébrique linéaire surdéterminé de dimension quelconque en calculant la solution des moindres carrés. La matrice **`M`** doit avoir plus de lignes que colonnes et le vecteur **`b`** doit avoir le même nombre de ligne que **`M`**.

La procédure **`CalculIntersection()`** permet de construire le système algébrique linéaire **`M a = b`**, puis il demande l'exécution de la fonction **`MoindresCarres(M,b)`** afin de calculer la solution **`a`**. Finalement, il imprime dans la fenêtre de débogage le contenu du vecteur solution **`a`**.

```
'Calcul la solution des moindres carrés du système algébrique: M a = b
Function MoindresCarres(M, b As Variant) As Variant
  Dim a(), Mt(), Z() As Variant        'Déclaration des matrices et vecteurs

  Mt = WorksheetFunction.Transpose(M)  'Mt est la transposé de M
  Z = WorksheetFunction.MMult(Mt, M)   'Produit des matrices Mt par M
  Z = WorksheetFunction.MInverse(Z)    'Calcul l'inverse de Z
  Z = WorksheetFunction.MMult(Z, Mt)   'Produit de Z par Mt
  a = WorksheetFunction.MMult(Z, b)    'Solution de moindre carré a = inv(Mt M) Mt b
  MoindreCarre = a
End Function
```

```
'Calcul le point (x,y) d'intersection des droites: x=0; y=0; et x+y=1
Sub CalculIntersection()
  Dim M(), a(), b() As Variant  'Déclaration de la matrice et des vecteurs
  ReDim M(1 To 3, 1 To 2)       'M est une matrice de 3 lignes x 2 colonnes
  ReDim b(1 To 3, 1 To 1)       'b est un vecteur de 3 lignes x 1 colonne

  M(1,1)=1: M(1,2)=1: b(1,1)=1  'Assigner le contenu à M et b
  M(2,1)=0: M(2,2)=1: b(2,1)=0
  M(3,1)=1: M(3,2)=0: b(3,1)=0

  a = MoindresCarres(M,b)  'M doit avoir plus de lignes que de colonnes
  Debug.Print "Solution a = [" & a(1,1) & "; " & a(2,1) & "]"
End Sub   'ExempleMatrice.xlsm
```

```
Call CalculIntersection()
Solution a = [0,333333333333333; 0,33333333333333]
```

4.3 Rapport d'identification des pièces

Un script VBA doit être préparé afin d'automatiser la production des rapports de crédits aux clients pour le retour de pièces à reconditionner. Pour chaque lot, une machine effectue le trie des pièces et fournie un fichier texte contenant les pièces identifiées et leur quantités. Tel que montré à la Fig. E.15, le format de ce fichier est toujours le même. Les numéros de piéces sont toujours dans le même ordre dans la colonne 1 et les quantités dans la colonne 2. Cependant, une ligne est omise lorsqu'il n'y a pas de pièce de ce numéro. Ainsi, le nombre de ligne du fichier texte est variable. Le fichier **CreditRecup.xlsm** contient le script VBA `CreditRecup()` et le rapport de Crédit de récupération à compléter, dont le prix accordée pour chaque pièce usagée. L'objectif du script est de :

1. lire le fichier `RecupNoXXX.txt` fourni par la machine;
2. transférer à la colonne 2 les quantités de pièces reçues;
3. transférer à la cellule **D1** le nom du fichier texte;
4. créer un nom de fichier rapport à partir du nom du fichier traité
   en remplacant "Recup" par "Rapport" et l'extension ".txt" par ".xlsm"; et finalement,
5. sauvegarder le fichier rapport sous son nouveau nom `RapportNoXXX.xlsm` qui permet d'identifier le lot XXX.

Une macro VBA est enregistrée sous le nom `CreditRecup()` avec le raccourci clavier *Ctrl-Shift-A*. L'enregistrement débute par la sélection et l'ouverture d'un fichier `.txt`, la fermeture de ce fichier, et finalement, la sauvegarde du classeur en format `.xlsm`. Ces instructions serviront de base au script VBA final. Des variables de type chaînes de caractères sont définies pour les noms de fichiers. La variable `ws` de type `Worksheet` est définie pour désigner le classeur du rapport. L'instruction Set est utilisée pour désigner la feuille 1 du classeur actuel (le rapport). La fonction `InputBox()` est utilisée pour entrer le nom du fichier `.txt` à traiter. Une fois le fichier texte ouvert dans un nouveau classeur, le nom du fichier texte est inscrit à la cellule **D1** du rapport. Puisque le fichier texte peut contenir

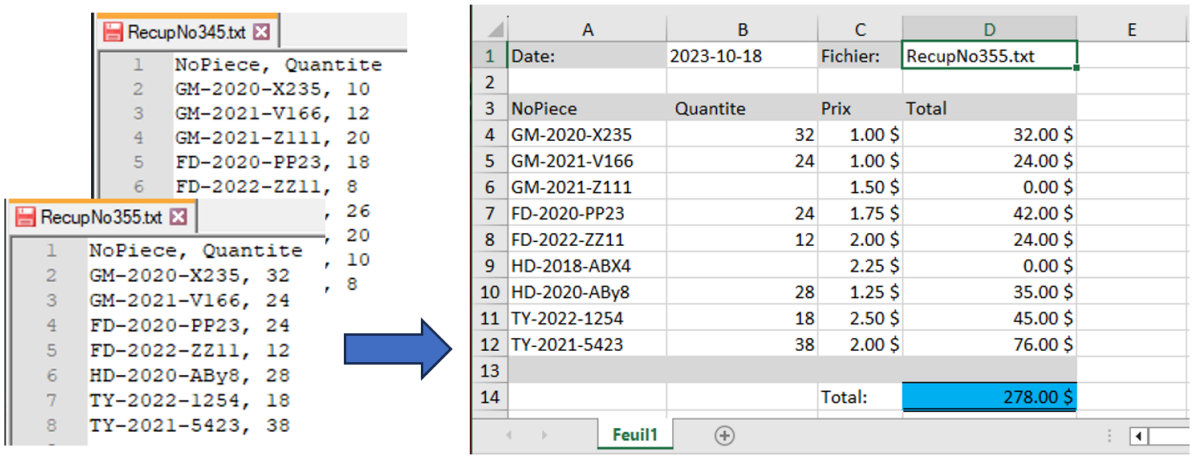


Fig. E.15 - Rapport d'identification des pièces au reconditionnement

seulement les pièces reçues, mais toujours dans le même ordre, deux indices sont utilisés. La boucle parcours le fichier rapport de la ligne 4 à 12 (variable `i`), alors que le fichier de texte est parcouru de la ligne 2 à la dernière ligne (variable `j`). Les quantités sont transférées pour chaque pièce ayant le même numéro de pièce. Le fichier texte est fermé sans aucune sauvegarde. La fonction `str = Replace(str1,str2,str3)` permet de remplacer dans `str1` la chaîne `str2` par la chaîne `str3`. Le nom du fichier texte de départ est modifié de **RecupNo355.txt** à **RapportNo355.xlsm**, puis sauvegardé dans le répertoire courant dans le format prenant en charge les macros. Plusieurs instructions Debug.Print permettent de suivre la progression de l'exécution du script, tel que montré dans la fenêtre de débogage.

```
Sub CreditRecup()
'CreditRecup Macro
'Touche de raccourci du clavier: Ctrl+Shift+A
  Dim NomFichier, NomRapport As String
  Dim ws As Worksheet
  Set ws = ThisWorkbook.Sheets("Feuil1")    'Feuille "Feuil1" du classeur CreditRecup.xlsm

  NomFichier = InputBox("Nom du fichier ?", "Pièces identifiées", "RecupNoXXX.txt")
  Debug.Print "Ouverture fichier : ", & NomFichier
  Workbooks.OpenText FileName:=NomFichier, _
   Origin:=xlMacintosh, StartRow:=1, DataType:=xlDelimited, TextQualifier _
   :=xlDoubleQuote, ConsecutiveDelimiter:=False, Tab:=True, Semicolon:= _
   False, Comma:=True, Space:=False, Other:=False, FieldInfo:=Array(Array( _
   1, 1), Array(2, 1)), TrailingMinusNumbers:=True 'Ouverture du fichier .txt

  ws.Cells(1, 4) = NomFichier       'Inscrire le nom du fichier dans le rapport
  j = 4: jFin = 12
  i = 2: iFin = Cells(Rows.Count, 1).End(xlUp).Row    'Cherche la dernière ligne des données
  Do
    If ws.Cells(j, 1) = Cells(i, 1) Then
      ws.Cells(j, 2) = Cells(i, 2)    'Inscrire la quantité dans le du rapport
      Debug.Print "Transfert ligne " & i & " -> " & j
      i = i + 1
    End If
    j = j + 1
  Loop Until i > iFin Or j > jFin
  ActiveWorkbook.Close SaveChanges:=False    'Fermer le fichier.txt
  NomRapport = Replace(NomFichier, ".txt", ".xlsm")
  NomRapport = Replace(NomRapport, "Recup", "Rapport")
  Debug.Print "Sauvegarde fichier : ", & NomRapport    'Sauvegarde du fichier Rapport .xlsm
  ThisWorkbook.SaveAs Filename:=NomRapport, FileFormat:= _
   xlOpenXMLWorkbookMacroEnabled, CreateBackup:=False
End Sub    'ExempleEE.xlsm
```

```
Ouverture fichier : RecupNo355.txt
Transfert ligne 2 -> 4
Transfert ligne 3 -> 5
Transfert ligne 4 -> 7
...
Sauvegarde fichier : RapportNo355.xlsm
```

4.4 Barre de suivi budgétaire

Un script VBA doit être préparé afin de faciliter le suivi budgétaire de votre organisation. Tel que montré à la Fig. E.14, les données budgétaires sont groupées en 3 colonnes : 1) une description du poste de dépenses; 2) le montant du budget alloué; et 3) le montant dépensé à ce jour. L'objectif est : 1) de souligner en jaune les montants de dépassement budgétaire; et 2) d'afficher une barre de suivi budgétaire dans la colonne de droite (colonne 4). Chaque *Barre de suivi budgétaire* est une chaîne de caractères de la fonte spéciale **Wingdings**. Le carré vide (lettre "o") est utilisé pour représenter le budget alloué, alors que le carrés plein (lettre "n") est utilisé pour représenter les dépenses réalisées à ce jour. Lorsqu'il y a dépassement budgétaire, les carrés pleins excédentaires sont affichés en rouge. Dans cette exemple, la valeur monétaire d'un carré est choisi à 1000$. La Fig. E.13 montre le fichier de suivi budgétaire après l'exécution du script VBA par un clic sur le bouton `Tirelire`.

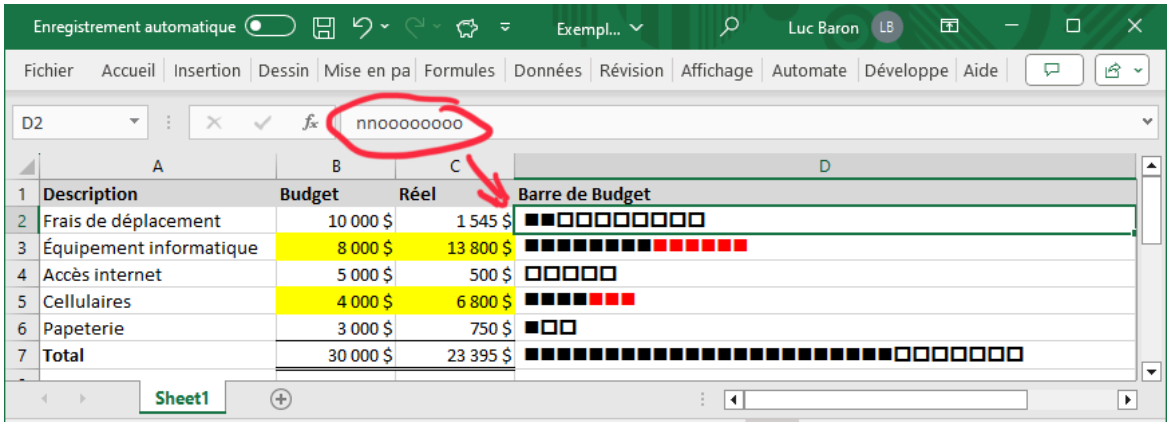


Fig. E.14 - Affichage d'une barre de suivi budgétaire

Une macro VBA est enregistrée sous le nom `Macro1()` avec le raccourci clavier *Ctrl-Shift-B* et aucune instruction. L'instruction `Call BarreBudget(2, 2, 1000)` est manuellement entrée avec l'éditeur VBA. Les paramètres d'entrées sont les numéros de ligne (2) et colonne (2) du 1er montant de la colonne budget et la valeur de 1000$ d'un carré. Les barres de suivis budgétaires sont affichées dans la 2e colonne à droite de la colonne budget.

```
Sub Macro1()
' Macro1 Macro relié au bouton Tirelire de la barre d'outil Accès rapide
' Touche de raccourci du Clavier: Ctrl+Shift+B
  Call BarreBudget(2, 2, 1000)   'Début du budget à la cellule (2,2) et Jeton à 1000$
End Sub

Function max(x,y As Long) As Long
  max = IIf(x > y, x, y)         'Retourne le maximum des deux valeurs
End Function

Sub BarreBudget(iDebut, j As Long, Jeton As Currency)
  Dim nBudget, nReel, nRestant As Long        'Nombre entier de jetons
  iFin = Cells(Rows.Count, j).End(xlUp).Row   'Cherche la dernière ligne du budget (colonne j)
  For i = iDebut To iFin                      'Pour toutes les lignes du budget
    With Cells(i, j)       'Pour la cellule (i,j) - Colonne budget alloué
      nBudget = Round(.Value / Jeton, 0)       'Budget alloué (nombre de jetons)
      nReel = Round(.Offset(0,1) / Jeton, 0)   'Dépenses à ce jour (nombre de jetons)
      nRestant = max(0, nBudget - nReel)       'Budget restant (nombre de jetons)
      If nReel > nBudget Then                  'S'il y a dépassement budgétaire
        .Resize(1,2).Interior.Color = vbYellow 'Remplissage en jaune
      Else
        .Resize(1,2).Interior.Color = xlNone   'Aucun remplissage
      End If
      With .Offset(0,2)    'Pour la cellule (i,j+2) - Colonne Barre de suivi budgétaire
        .Font.Bold = True
        .Font.Color = vbBlack
        .Font.Name = "Wingdings"     'Fonte spéciale: n=carré plein; o=carré vide
        .Value = String(nReel, "n") & String(nRestant, "o")
        If nReel > nBudget Then      'Si le réel dépasse le budget
          .Characters(nBudget + 1).Font.Color = vbRed   'Affiche le dépassement en rouge
        End If
      End With
    End With
  Next i
End Sub   'ExempleED.xlsm
```

La procédure **`BarreBudget`** utilise la fonction **`max(x, y)`** qui retourne la valeur entière maximale entre les deux valeurs fournies. Elle débute par déterminer **`iFin`** la dernière ligne de données de la colonne 2 par une méthode robuste. Une boucle **`For`** est utilisée pour traiter chacune des lignes du budget de **`iDebut`** à **`iFin`**. Pour chaque ligne, on récupère dans **`nBudget`** le budget alloué de la cellule (i,j) et dans **`nReel`** les dépenses réelles à ce jour de la cellule (i,j+1), divisés par la valeur d'un jeton (variable **`Jeton`**) et arrondis au nombre entier le plus près. Le nombre de jetons restant **`nRestant`** est la différence entre le budget et le réel. C'est la disponibilité budgétaire. Si **`nReel > nBudget`**, il y a dépassement budgétaire et on fait le remplissage en jaune des cellules budget et réel, sinon on annule le remplissage existant. Dans la 4e colonne, la couleur de caractères noir est appliquée, la fonte gras est activée, ainsi que la fonte spéciale **Wingdings**. La fonction **`String(nReel, "n")`** permet de construire une chaîne de **`nReel`** fois le caractère "n", comme montré à la cellule **D2** de la Fig. E.13. S'il y a dépassement budgétaire, alors les carrés exédentaires sont affichés en rouge. La méthode **`.Characters(n)`** permet de modifier les propriétés à partir du n*ième* caractère jusqu'à la fin de la chaîne. Le script doit être réexécuté à chaque modification des montants du budget.

5 Conclusions

La programmation de script VBA pour l'automatisation de tâches industrielles et commerciales est un moyen sûre, flexible et efficace pour traiter et formatter un volume moyen de données en provenance et à destination de systèmes de gestion de l'entreprise. L'utilisation généralisée de la suite Office rends l'utilisation de cet outil encore plus pertinent.

Copyright 2024 Polytechnique Montréal par Luc Baron