Alexia Ronsavoir

isaralyon

Une école d'ingénieurs au coeur de la vie

Macro-commandes EXCEL 2007

Séquence d'opérations automatisées

Jean GALLICE

2013

1) Afficher le Ruban Développeur d'Excel 2007

Bouton Office ⇨ **Options Excel** ⇨ **Standard** ⇨ **Afficher l'onglet Développeur dans le ruban**

1- La zone **Code** contient les commandes nécessaires aux macros VBA classiques

2- La zone **Contrôles** permet d'insérer des objets (zones de listes…) dans une feuille de calcul, de créer des boîtes de dialogue personnalisées et de visualiser leurs propriétés

2) Méthode simple mais limitée : l'enregistrement d'une macro

Ruban Développeur groupe **Code** ⇨ **Enregistrer une macro…** ou icône en bas à gauche de l'écran (si elle est absente : clic droit sur la Barre d'état, activer **Enregistrement de macro**)

Dans **"Ce classeur"** : s'exécute sur le classeur en cours et est enregistrée avec ce classeur

Dans le "**Classeur de macros personnelles**" : utilisable dans tous les classeurs

Attention : le Classeur de macros personnelles est **masqué**. Pour modifier les macros, on doit l'afficher par le **Ruban Affichage** groupe **Fenêtre** ⇨ **Afficher…** Il s'appelle **PERSONAL.XLSB** et se trouve dans le dossier **C:\Utilisateurs***Login*\ **AppData\Roaming\Microsoft\Excel\XLSTART** où *Login* est votre nom de connexion.

Utiliser les références relatives (ou absolues) : icône ?

Relatives (icône enfoncée) : exécute les opérations à partir de la cellule où l'on est placé au départ de la macro et enregistre les déplacements effectués

Absolues (icône normale) : quelque soit la cellule sélectionnée au départ, exécute les opérations sur les cellules définies lors de l'enregistrement

Arrêter l'enregistrement avec du groupe **Code** ou l'icône située tout en bas à gauche de l'écran

3) Relire ou Exécuter une macro

a) Ruban Développeur groupe **Code** ⇨ **Macros** ou Alt **F8** puis **Modifier** ou **Exécuter**

b) *Arrêter l'exécution d'une macro* (en cas de "plantage" ou de "boucle sans fin"…)

1- Appuyer sur la touche **Esc** (ou **Echap**) et cliquer sur **Débogage** pour essayer de repérer l'erreur
⇨ Voir le Mode Arrêt ("ligne jaune") en page 4 paragraphe 5)g)

2- Repasser dans VBA et cliquer si possible sur l'icône

3- Si cela ne suffit pas : arrêter Excel avec **Ctrl Alt Suppr** (toujours enregistrer avant !)

c) *Bouton Macro dans une feuille :* utiliser l'icône
Ruban Développeur groupe **Contrôles** ⇨ **Insérer** ⇨ **Contrôles de Formulaire** (pas **ActiveX** !)

Dessiner le bouton avec la souris (+ Alt pour utiliser le quadrillage des cellules)

1- *Sélectionner/Modifier un bouton* — Clic droit, puis **Echap**

Effacer un bouton — ou Ctrl Clic **Suppr**

2- *Déplacer, Redimensionner* — et (+ Alt pour utiliser le quadrillage)

d) *Icône personnelle dans la barre d'outils Accès rapide* (utilisable dans tous les classeurs)

Clic droit sur n'importe quelle icône ⇨ **Personnaliser la barre d'outils Accès rapide**
⇨ catégorie **Macros** ⇨ choisir la macro à lier ⇨ **Ajouter**, puis **Modifier…** pour changer l'image

Supprimer l'icône : clic droit dessus, **Supprimer de la barre Accès rapide**

4) Sécurité des macros

Ruban Développeur groupe **Code** ⇨ **Sécurité des macros** ⇨ **Désactiver les macros avec notification**
Lors de l'ouverture du classeur, Excel propose un bouton d'**Activer les macros** ou **Options**
Présence d'un bouton **Options** sous le ruban dans certaines configurations

5) Saisie d'une macro-commande VBA

a) *Objets, propriétés, méthodes et événements*

1- *Objet*

Un **objet** représente un *élément d'une application*, tel qu'une feuille de calcul, une cellule ou un graphique.
Il faut identifier un objet avant de pouvoir appliquer l'une des **méthodes** (procédure agissant sur un objet) ou modifier la valeur de l'une de ses **propriétés** (taille, couleur, emplacement à l'écran ou état, par exemple activé ou désactivé).

Objets les plus courants :

- ➲ **Application** (Excel)
  - ➔ **Workbook (classeur)**
    - ↳ **Worksheet (feuille de calcul)**
      - ⇨ **Names** (noms définis dans la feuille)
      - ⇨ **Range** (plage ou domaine)
        - → **Area** (blocs de sélection multiple)
        - → **Border** (bordure)
        - → **Font** (police)
        - → **Interior** (intérieur)
        - → **Name** (nom du domaine)
      - ⇨ **PageSetup** (mise en page)
      - ⇨ **ChartObject** (graphique incorporé)
        - → **Border** (bordure)
        - → **Chart** (graphique)
          - **ChartArea** (zone de graphique)
          - **ChartPlot** (zone de traçage)
          - **PageSetup** (mise en page)
          - **ChartTitle** (titre)
          - **SeriesCollection** (ensemble des séries)
          - **Axes**
          - **Legend**
          - ....
        - → **Interior** (intérieur)
      - ⇨ **Shape** (dessin dans la feuille de calcul)
    - ↳ **Chart** (feuille graphique)
      - ⇨ idem que **Chart** de **ChartObject**
    - ↳ **DocumentProperties** (propriétés du classeur, en lecture seule)
      - ⇨ Ex: auteur, nombre de pages, date de création…
    - ↳ **Windows** (fenêtres du classeur)
  - ➔ **Dialogs** (boites de dialogue d'Excel)
  - ➔ **CommandBars** (barres d'outils d'Excel)
  - ➔ ....

2- *Collection*

Une **collection** est un *objet contenant plusieurs autres objets*, généralement, mais pas toujours, du même type. Par exemple, l'objet **Worksheets** contient tous les objets **Worksheet**.

Les éléments d'une collection peuvent être identifiés par numéro ou par nom. Par exemple, dans la procédure suivante, **Workbooks(1)** identifie le premier objet **Workbook** ouvert.

```
Sub FermerClasseur()
    Workbooks(1).Close
End Sub
```

3- *Procédure*

Une procédure est une séquence nommée d'instructions exécutée en tant qu'entité. Par exemple, **Function**, **Property** et **Sub** sont des types de procédures. Un nom de procédure est toujours défini au niveau module. Tout code exécutable doit être contenu dans une procédure. Les procédures ne peuvent pas être imbriquées au sein d'autres procédures.

4- *Méthode*

Une **méthode** est une *action qu'un objet peut exécuter*. Par exemple, **Add** est une méthode de l'objet **ComboBox** qui ajoute une nouvelle entrée à une liste modifiable.

La procédure suivante utilise la méthode **Add** pour ajouter un nouvel élément à **ComboBox** :

```
Sub AjoutEntrée(newEntrée as String)
    Combo1.Add newEntrée
End Sub
```

5- *Propriété*

Une **propriété** est un *attribut d'un objet* définissant l'une des caractéristiques de l'objet telle que la taille, la couleur ou la position à l'écran, ou un aspect de son comportement, par exemple s'il est activé ou visible. Pour changer les caractéristiques d'un objet, changez les valeurs de ses propriétés.

Pour définir la valeur d'une propriété, faites suivre la référence d'un objet d'un point, du nom de la propriété, d'un signe égal (=) et de la nouvelle valeur de propriété. Par exemple, la procédure suivante masque la feuille 1 du classeur en définissant la propriété **Visible** :

```
Sub CacherOnglet()
    Worksheets(1).Visible = False
End Sub
```

Certaines propriétés ne peuvent pas être définies. La rubrique d'aide de chaque propriété indique si vous pouvez la définir (lecture-écriture), seulement la lire (lecture seule) ou seulement y écrire (écriture seule).

Vous pouvez extraire des informations sur un objet en renvoyant la valeur de l'une de ses propriétés. La procédure suivante utilise un message pour afficher le numéro de la couleur de la cellule active :

```
Sub AfficheNumCouleur()
    Couleur = Activecell.Interior.Colorindex
    MsgBox Couleur
End Sub
```

6- *Évènement*

Un **évènement** est une action reconnue par un objet, telle qu'un *clic de la souris* ou la *frappe d'une touche*, et pour laquelle vous pouvez écrire un code de réponse. Les événements peuvent être déclenchés par une action de l'utilisateur, par le code d'un programme ou par le système.

b) *Démarrer la saisie d'une macro*

☺ **Ruban Développeur** groupe **Code** ⇨ **Visual Basic** ou Alt **F11** Puis **Insertion… Module**

c) *Conseil pratique*

Si possible, plutôt que de saisir toutes les lignes d'une macro, **enregistrer** la séquence à utiliser (mise en gras, modification du format…) et faire un **copier/coller** dans votre macro en passant par **Fenêtre**

d) *Aide à la saisie*

Durant la saisie, un mot peut apparaître automatiquement dans une boite à liste :

- Pour le garder et aller à la ligne suivante : 
- Pour le garder et rester sur la ligne en cours : 

yeah

e) *Copie de mots ou de lignes déjà saisies ailleurs*

Sélectionner mot (double-clic) ou ligne (clic dans la marge) et utiliser le "glisser-copier" avec 

f) *Déclaration des variables*

Dans une macro, si aucune des variables n'est déclarée, elles sont automatiquement de type **Variant** et peuvent contenir n'importe quoi (chiffre, texte…). Satisfaisant pour les macros très courtes.

Cependant, déclarer les variables en début de macro permet :

- d'éviter les *fautes de frappe* dans le nom d'une variable (à condition de mettre **Option Explicit** avant la première macro)
- d'empêcher la saisie, par exemple, d'un texte dans une variable numérique
- de limiter l'*espace mémoire* utilisé lorsqu'on a de nombreuses variables

Types de variables (autre que **Variant** qui occupe de 16 à 22 octets) :

- **Byte** (1 octet) : 0 à 255
- **Boolean** (2 octets) : vrai ou faux (True/False)
- **Integer** (2 octets) : nombre entier de ..... -32 768 à +32 767
- **Long** (4 octets) : nombre entier de ..... -2 147 483 648 à +2 147 483 647
- **Single** (4 octets) : nombre décimal de ..... $-3{,}402823\ 10^{38}$ à $+3{,}402823\ 10^{38}$
- **Double** (8 octets) : décimal de..... $-1{,}79769313486231\ 10^{308}$ à $+1{,}79769313486232\ 10^{308}$
- **Date** (8 octets) : date et/ou heure ⇨ de -657 434,000006 (01/01/100 00:00:01) ***!! texte !!***
  à +2 958 465,99999 (31/12/9999 23:59:59)
- **String** (0 à 63 octets) : texte (chaîne de lettres et/ou nombres, espaces, ponctuations)
- **Object** (4 octets) : permet d'affecter une référence d'objet (Range, Selection, Worksheet…) à une variable (éventuellement précisé avec l'instruction **Set**)

Déclarer *avant* ou *après* **Sub** (niveau **Module** ou niveau **Procédure**) :

Les variables déclarées avant les macros (*avant* Sub) sont utilisables dans toutes les macros du Module. Déclarées *après* Sub (dans la macro), elles ne sont reconnues que dans cette macro.

Syntaxe :

Dim rang, toto **As Integer**, lig **As Byte**, col **As Byte**

Attention : ici, la variable *rang* est de type Variant si on ne précise rien !

g) *Mode Arrêt : la "ligne jaune"*

Si l'on choisi **Débogage** en cas d'erreur, la ligne incriminée passe en jaune : l'annuler en cliquant sur 

h) *Aide en ligne*

On peut cliquer sur un mot déjà saisi et appuyer sur la touche **F1**

Pour retrouver la syntaxe de tous les objets, propriétés, méthodes et événement, cliquer sur l'**Explorateur d'objets**  , choisir **Excel** dans la boite **<Toutes bibliothèques>**, cliquer sur l'objet recherché (à gauche), puis sur la propriété, la méthode ou l'événement voulu (à droite) et enfin sur 

La recherche dans l'aide générale d'Excel  est assez laborieuse et doit utiliser les liens proposés dans **Voir aussi** ou **Exemples**

6) Commandes de base pour les applications techniques

a) *Organisation*

```
Sub toto ()

End Sub
```

Possibilité d'utiliser un
`'` commentaire (apostrophe en début de ligne)

b) *Déclaration des variables* ⇨ ***voir b) page 5***

① `Dim toto As Integer` ② `Option Explicit` ⇨ contrôle de la saisie

③ `Dim Trame As Object` puis `Set Trame = Range("A1:C5")`

c) *Passage : Variable "informatique" ⇔ Cellule Excel*

① `S = Cells(3, 1).Value` ou : `S = Cells(lig, col).Value` avec `lig = 3` `col = 1`

② `Cells(1, 10).Value = X` ou `...="bonjour"` ou `...= 3` ou `...= """"`

Il faut doubler les guillemets

d) *Boucle simple*

```
For col = 1 To 10
  ---
  ---
Next col
```

e) *Tests*

① `If t > s Then` *opération* plus éventuellement `:` *autre opération*
évite le saut à la ligne

② `If t>s Then` *opération* `Else` *autre opération*

③
```
If t>s Then
   opération
   ---
   Else: autre opération
   autre opération
   ---
End If
```

Ex :
```
If v>0 And v<14 Then
    A=1
Else
    A=0
End If
```

f) *GoTo et End* *(éviter GoTo : logique trop confuse)*

①
```
GoTo 10

10
```
ou
```
10

GoTo 10
```
②
```
GoTo toto

toto:
```
③ `End` (arrêt de la macro)

g) *Sélections et plages*

① `Range("E5").Select` ou : `Cells(5, 5).Select`

② `Range("A1:C4").Select` ou : `Range(Cells(1, 1), Cells(4, 3)).Select`
↳ de A1 *jusqu'à* C4

différent de : `Range("A1,C4").Select` ⇨ A1 *et* C4

③ Avec *i=2* et *j=4* `Cells(i, j).Value = x`

Effacement d'une sélection : ④ `Selection.ClearContents`
(vider le contenu)

Sans sélection :
`Range("E5").ClearContents`

h) *Aspect d'une cellule*

```
Cells(1, 8).Interior.ColorIndex = 3

Selection.NumberFormat = "0.00"
                    ou = "d/m/yyyy"
                    ou = "h:m:ss"
Cells(5, 5).Value = Format(Var, "0.00")
```

couleurs : 0Auto 1N 2Bc 3R 4Vr 5B 6J 7Vi 8Bclair (<57)
Utiliser l'enregistrement auto. pour trouver le code
Attention : Auto vaut ***-4142*** en enregistrement !

7) Commandes d'usage courant pour les applications techniques

a) *Écriture d'une instruction longue sur plusieurs lignes*

```
Range("A15").Value = "Attention : la cantine sera fermée _
                                  exceptionnellement vendredi prochain."
```

espace + tiret bas

b) *Compteurs et cumuls*

① `Val = Val + 1` ② `Tot = Tot + x`

c) *Cellule active*

Après une sélection, attention à la notion de "cellule active", toujours unique :

① `ActiveCell.Value = Cells(10, 3).Value` ou `ActiveCell.Value = "bonjour"`

② `ActiveCell.ClearContents` différent de : `Selection.ClearContents`

③ `If ActiveCell.Value > 2 Then` *opération...*

④ `Range("C4").Activate` Dans une sélection, détermine la cellule "blanche"

⑤ Repérage du n° de ligne et du n° de colonne de la cellule active

```
lig = ActiveCell.Row : col = ActiveCell.Column
If lig >= 3 And Lig <= 8 And col >= 5 And col <= 9 Then ...
```

Vérification d'appartenance à une zone

d) *Opérations sur une sélection ou en proximité*

① Suppression d'une sélection : `Selection.Delete Shift:=xlUp`
(**détruire**) ou : `Shift:=xlToLeft`

② Sélection d'une ligne ou d'une colonne

`Rows("6:6").Select` `Columns("D:E").Select`

③ Décaler la sélection : `ActiveCell.Offset(1, 0).Select` (1 ligne vers le bas, même colonne)

④ Remplir une cellule sans la sélectionner :

`Cells(lig, 2).Offset(1, -1).Value = x` (une ligne vers le bas, une colonne à gauche)

e) *Structures de boucles*

① `val = 0` (valeur de départ pour test en tête de boucle)

```
Do Until val >= 9          ou :   Do While val < 9
  ...
  val = formule de calcul
  DoEvents
Loop
```

Placer un **DoEvents** dans les boucles pour permettre la sortie d'urgence

②
```
Do
  ...
  val = formule de calcul
  DoEvents
Loop Until val >= 9 ou : Loop While val < 9
```

③
```
Do Until...
  ...
  Exit Do
  ...
Loop
```
sortie sous le `Loop`

④ Balayage automatique d'une sélection :

```
For Each z In Selection
   z.Value = z.Value + 1
Next z
```

`z` doit être déclaré en `Object`

`z` étant ici assimilé à une cellule, il doit s'utiliser avec les propriétés correspondantes

f) *Utilisation d'une formule Excel dans une procédure VBA*

```
ActiveCell.Value = Application.WorksheetFunction.Sum(Range("A2:A9"))
```

On utilise le mot anglais de la fonction Excel : MOYENNE=**AVERAGE**, NB=**COUNT**

Par contre, les domaines concernés par la fonction utilisent la codification VBA :

```
Cells(i, 3).Value = Application.WorksheetFunction.Sum(Range(Cells(i, 1),Cells(i, 5)))
```

VBA dispose également en interne de quelques fonctions classiques propres à tout langage de programmation :

`N=Int(Rnd*10000)` génère un nb entier aléatoire de 0 à 9999

g) *Classeurs et feuilles*

① Ouvrir, fermer, enregistrer ou non un classeur

```
Workbooks.Open FileName:="A:\nomfichier.xls"
```
Utiliser **OpenText** si c'est un fichier txt

```
ActiveWindow.Close savechanges:=True
```
ou **False**

```
ActiveWorkbook.Save
```

```
ActiveWorkbook.SaveAs FileName:="C:\Dossier\nomfichier.xls"
```

② Création d'un nouveau classeur vide dans une même session d'Excel

```
Workbooks.Add
```

③ Création d'une feuille supplémentaire dans le classeur actif

```
Sheets.Add
```

④ Suppression d'une feuille dans le classeur actif :

```
SendKeys "{ENTER}"
Sheets(1).Delete
```
permet de valider automatiquement la boite de demande de confirmation

⑤ Renommer un onglet (une feuille) :

```
Sheets("Feuil1").Name = "titi"
```

⑥ Activation d'un classeur ou Activation/Sélection d'une feuille

```
Windows("Classeur2.xls").Activate
```
(Le classeur doit être déjà ouvert)

```
Workbooks("ClasBilan.xls").Activate
```

```
Sheets("Feuil3").Select
```
ou : `Sheets(3).Activate`

Basculer sur le classeur ouvert "suivant"

```
ActiveWindow.ActivateNext
```

⑦ Adresse complète d'une cellule (et "copie rapide" d'un contenu)

On peut attribuer le nom d'un classeur à une variable : toto = "ClasBilan.xls"

```
Workbooks("Classeur1.xls").Sheets("Feuil3").Cells(5, 2).Value =
                                Workbooks(toto).Sheets(1).Cells(1, 3).Value
```

⑧ Compter le nombre de feuilles dans le classeur actif

```
nb = Sheets.Count
```

⑨ Compter le nombre de classeurs ouverts dans la même session Excel

```
x = Workbooks.Count
```

⑩ Protéger/Déprotéger une feuille

Sur une feuille protégée (Outils Protection), il faut déprotéger juste avant de modifier et reprotéger ensuite

```
ActiveSheet.Unprotect
```

```
ActiveSheet.Protect
```

h) *Acquisition de données*

① Temporisation interne (temps "infime")

```
For i = 1 To 10 000 000 : Next i
```
(*attendre* pendant la durée de la boucle : dépend du processeur)

② Temporisation avec délai fixé (`Timer` : nb de secondes écoulées depuis minuit)

```
gong = Timer + 1.5
Do Until Timer >= gong
  DoEvents
Loop
```
gong prend la valeur Timer + délai **souhaité**
"Tourner" jusqu'à ce que Timer atteigne la valeur prévue

Attention : remise à zéro à minuit !

③ Récupération de l'heure courante

```
Cells(1 , 3).Value = Now() ou Time()
```
utiliser un format adapté : "h:mm:ss" ou "dd:mm:yy"

④ Extraire la seconde de l'heure courante

```
y = Second(Time)
```

⑤ Fonctions "texte" (soit toto une variable déclarée en `String`)

`Len(toto)` donne le nombre de caractères de la variable

`Mid(toto,4,2)` extrait de toto les deux caractères à partir du 4ème

`Right(toto,2)` extrait de toto deux caractères à partir de la droite. `Left` pour la gauche

⑥ Imprimer au fil de l'eau

En début de procédure :
```
Open "lpt1" For Output As #1
Print #1, " Heure " + VbTab + "Temp"
Print #1,
Close #1
```
`lpt1` (ou LPT1) est la sortie imprimante
Utiliser une imprimante à aiguilles (sinon : une page par ligne !)

Dans la boucle d'acquisition :
```
Open "lpt1" For Output As #1
Print #1, Format(Now, "hh:mm") + VbTab + Format(t, "0.00")
Close #1
```

⑦ Création d'un fichier des données sur support informatique

En début de procédure :
```
Open "a:\aquisauv.xls" For Output As #1
Seek #1, 1
Close #1

Open "a:\aquisauv.xls" For Append As #1
Print #1, "Heure" + vbTab + "Debit"
Close #1
```

Dans la boucle d'acquisition :
```
Open "a:\aquisauv.xls" For Append As #1
Print #1, Format(Time, "hh:mm:ss") + vbTab + Format(débit, "0.00")
Close #1
```

⑧ Création d'un "graphique glissant" *voir annexe de TP*

i) *Messages et boîtes de saisie*

① Message simple : informer l'utilisateur

`MsgBox "bonjour"` → affiche une boite de dialogue avec le mot bonjour

Soit `cont = 150`

`MsgBox cont` → affiche une boite de dialogue avec 150 (éviter "cont")

② Boite MsgBox avec gestion des boutons (OK, Annuler, Aide…) et des icônes

`x = MsgBox("message",0,"titre de la boite")`

0	Affiche le bouton OK uniquement
1	Affiche les boutons OK et Annuler
2	Affiche le bouton Abandon, Réessayer et Ignorer
3	Affiche les boutons Oui, Non et Annuler
4	Affiche les boutons Oui et Non
5	Affiche les boutons Réessayer et Annuler
Additionner un des chiffres ci-dessus avec :	
16	Affiche l'icône *Message critique*
32	⇨ l'icône *Requête d'avertissement*
48	⇨ l'icône *Message d'avertissement*
64	⇨ l'icône *Message d'information*

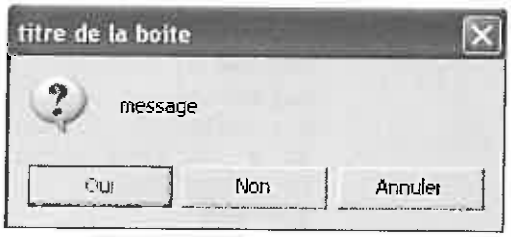
Valeurs renvoyées :

vbOK	1	OK
vbCancel	2	Annuler
vbAbort	3	Abandonner
vbRetry	4	Réessayer
vbIgnore	5	Ignorer
vbYes	6	Oui
vbNo	7	Non

Exemple :

`x = MsgBox("message",35,"titre de la boite")`

donne :



Si on clique sur Non on obtient `x = 7` ou `x = vbNo`

Si on clique sur Annuler on obtient `x = 2` ou `x = vbCancel` ⇨ utiliser un **`If`**

③ Boite de saisie

`x = Application.InputBox("message", "titre", "10", , , , , 1)`

valeur affichée par défaut

1 → numérique
2 → texte
3 → indifférent

Exemples de tests pouvant suivre un "InputBox" :

`If x = False Then End`	si on a cliqué sur "Annuler" : arrêt de la macro
`If x="" Then…`	si on a laissé vide
`If IsNumeric(x)=False Then…`	pour vérifier si x est numérique
`If x<1 Or x>7 Then…`	pour vérifier par rapport à des bornes
`x = Int(x)`	si on attend un nombre entier (partie entière)

④ Améliorer les messages dans les boites de dialogue

Ceci s'utilise dans la zone *"message"* d'une **`InputBox`** ou d'une **`MsgBox`**

Insérer un saut de ligne dans le message : `("première partie"+ vbCr +"deuxième partie",…`

ou : `+ Chr(10) +`

Associer un texte et une variable : `("message " & x,…` où x peut être numérique ou texte

8) Faire face à quelques situations particulières

a) *Blocage de l'affichage :* *évite les déplacements à l'écran pendant l'exécution*

`Application.ScreenUpdating = False` `= True` pour débloquer

b) *Positionnement d'une cellule donnée (ici : E5) en haut à gauche de l'écran*

```
Application.Goto reference:=Range("E5"), Scroll:=True
```

c) *Gestion des macros trop volumineuses : appeler différentes macros à partir d'une procédure principale*

① Appel d'une autre procédure (appelée Toto) :

```
.....
.....
Toto
.....
```

quand la procédure **Toto** est terminée retour automatique à la ligne suivante

② Sortie anticipée de la procédure appelée : `Exit Sub` revient à la macro appelante

③ Arrêt anticipé de <u>toutes</u> les procédures : `End`

④ Passage de paramètres entre procédure appelante et procédure appelée :
Les variables d'une macro restent inconnues des autres macros : c'est la raison d'être des parenthèses de Sub.
On peut passer des variables, des constantes, des cellules Excel :

```
Sub Tata()
x = "bonjour"
fluct = 10
Cells(1, 3).Value = 150
Tati x, 2, numer, fluct
MsgBox numer
MsgBox fluct
End Sub
```

```
Sub Tati (k, m, koder, x)
koder = 0
MsgBox k
MsgBox m
z = Cells(1, 3).Value
If z > 2 Then koder = 1
If m = 2 Then x = -x
End Sub
```

ATTENTION : seul compte <u>l'ordre</u> des variables transférées ! Ici k=x m=2 numer=koder et fluct=x !
Donc le x "interne" à Tata n'a rien à voir avec le x "interne" à Tati !
Préférer des noms de variables cohérents…

d) *Démarrer automatiquement une macro à l'ouverture ou à la fermeture du classeur*

Le nom à utiliser pour la macro est
① `Sub Auto_open()` pour l'ouverture ② `Sub Auto_close()` pour la fermeture

e) *Appel d'une boîte de dialogue Excel*

`Application.Dialogs(5).Show` pour la boîte "Enregistrer sous" (ou **xlDialogSaveAs**)
1 (un) pour la boîte "Ouvrir" (ou **xlDialogOpen**)
7 pour la boîte "Mise en page" (ou **xlDialogPageSetup**)
(utiliser le mot-clé plutôt que les n°)

f) *Génération de nombres aléatoires*

Randomize évite de générer des séries répétitives : **Rnd** calcule un nombre aléatoire entre 1 et 0.

```
n1 = Int(Rnd * 255)
n2 = Int(Rnd * 255)
n3 = Int(Rnd * 255)
teinte = RGB(n1, n2, n3)
Cells(5, 1).Interior.Color = teinte
```

ne pas confondre avec `ColorIndex`

g) *Sélectionner la totalité d'un tableau n'ayant pas de lignes vides*

Sélectionner une cellule du tableau, puis : `Selection.CurrentRegion.Select`

h) *Aller à la fin d'un tableau n'ayant pas de lignes/colonnes vides*

```
Cells(1, 1).End(xlDown).Select
```

```
ActiveCell.End(xlToRight).Select
```

i) *Trouver la dernière ligne d'une colonne (ici 1) même si elle présente des cellules vides*

```
Cells(Rows.Count, 1).End(xlUp).Select
```

j) *Insérer une formule dans une cellule d'Excel (rare car il est plus facile de faire le calcul dans VBA)*

① `ActiveCell.Value = "=SUM(R[-10]C:R[-1]C)"` ⇨ =SOMME(L(-10)C:L(-1)C)

② `ActiveCell.Value = "=INDEX(Feuil2!R2C1:R11C1,Feuil1!R[-1]C)"`

k) *Procédure "Function"*

Création d'une fonction personnelle avec VBA

Exemple : Pour créer une fonction calculant automatiquement le prix TTC en fonction du prix HT et du taux de TVA :

```
Function TTC(HT, TTVA) As Double
TTC = HT + (HT * TTVA / 100)
End Function
```

Utilisable dans une cellule Excel :

=TTC(B3;B4) où B3 contient le *prix HT* et B4 le *taux de TVA*

=**SI**(TTC(B3;B4)>1000;"Vérifier";"OK")