VBA Excel

par Michel Berthiaume

http://www.usherbrooke.ca/adm/departements/simqg/professeurs/michel-berthiaume/

Introduction

Sur cette page...

Les prochaines pages...

Introduction

Visual Basic for Application est un environnement de programmation qui accompagne et permet d'automatiser la plupart des applications bureautiques de Microsoft.

On peut trouver une description générale du langage sur Wikipedia.

Dans l'environnement Excel, VBA sert à programmer des macro-commandes (ou macros), ce qui est un autre nom pour des programmes. Dans le présent tutoriel, on évitera le terme macro-commandes (ou macros), sauf lorsque Microsoft utilise ce terme.

Connaissances préalables

La compréhension de ce qui suit requiert une connaissance fonctionnelle de Excel.
On peut trouver un tutoriel sur Excel 2007 ici : Le compagnon Info Excel.

Configurer Excel

L'installation par défaut d'Excel ne donne pas facilement accès aux outils de développement et d'exécution de programmes VBA.

Consultez les documents ci-dessous pour configurer Excel:

Afficher l'onglet DÉVELOPPEUR

Activer l'exécution des macros

De plus, le format d'enregistrement par défaut des classeurs Excel exclut les macros VBA. Il faut donc faire attention lorsqu'on enregistre un classeur Excel contenant du code VBA:

Enregistrer un classeur Excel contenant une macro VBA

Premier exemple:
Identifier par une couleur les cellules déverrouillées de la feuille de travail active

Ce premier exemple illustre l'utilisation de VBA pour automatiser une tâche Excel qu'il serait fastidieux de faire manuellement:
Colorer en jaune chaque cellule déverrouillée.

Démarrez Excel avec un classeur vierge.

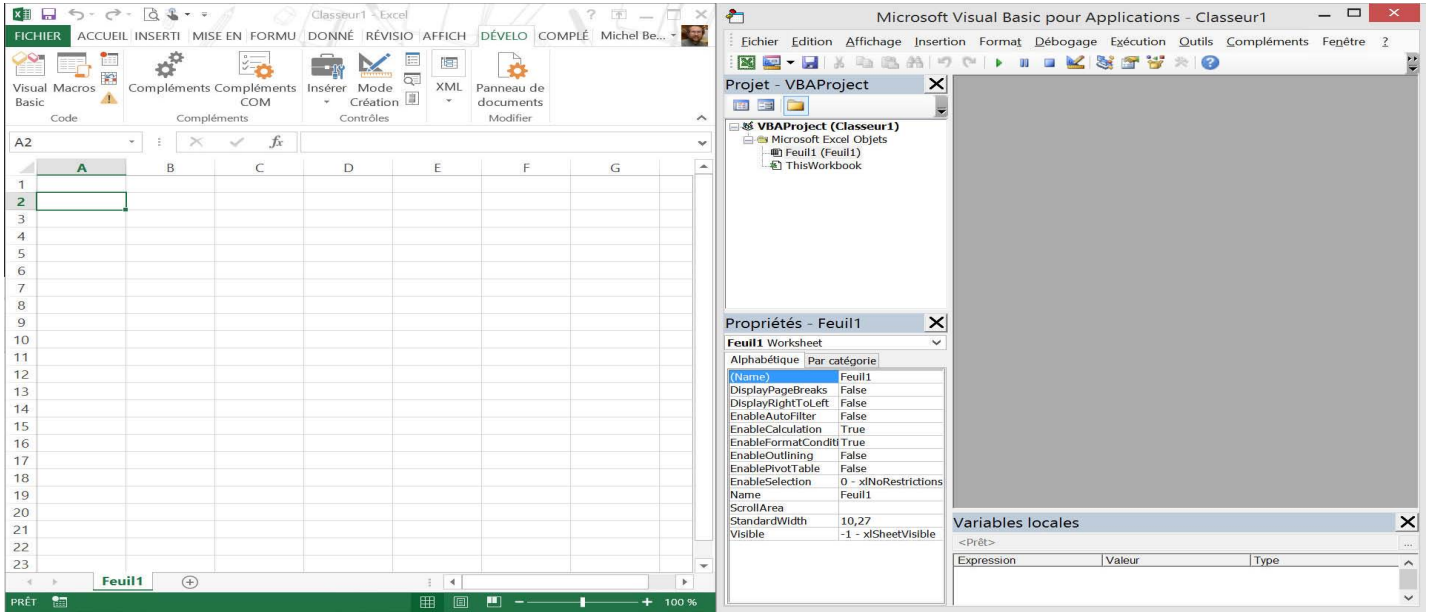
Déverrouillez quelques cellules (A2, B1 et D1 par exemple)

Déverrouiller des cellules Excel

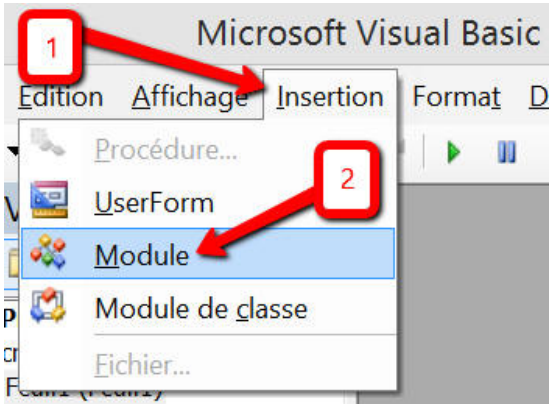
Ouvrez l'onglet DÉVELOPPEUR, puis cliquez le bouton Visual Basic:

Si l'onglet DÉVELOPPEUR n'est pas affiché, retournez consulter comment configurer Excel.

Suggestion: disposez les 2 fenêtres (Excel et VBA) côte à côte:

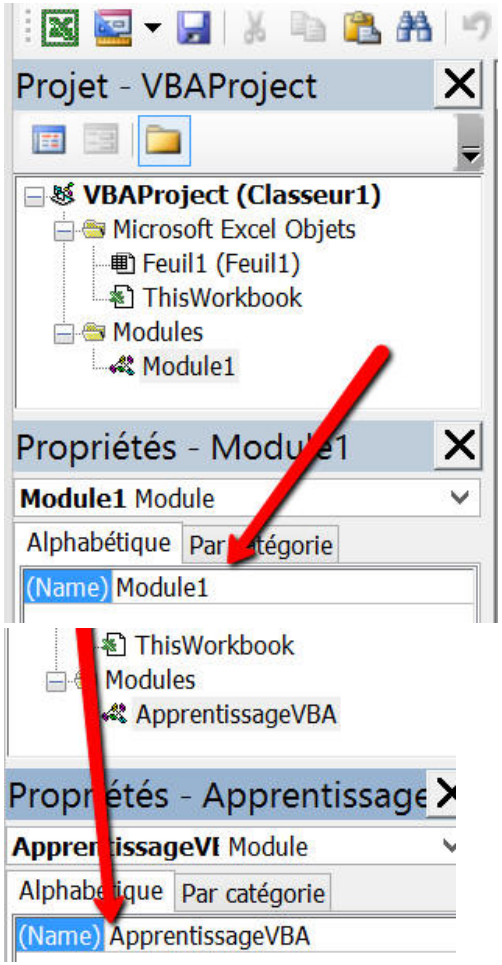


Du menu **Insertion** (de VBA), sélectionnez **module**:



Changez le nom du module dans la zone de texte (**Name**) en bas à gauche. Remplacez **Module1**

par un nom significatif ne contenant pas d'espace
ApprentissageVBA
par exemple.



Tapez (ou copiez-collez!) ce code VBA:

```
Sub ArrièrePlan()
'Auteur: Michel Berthiaume
'Mettre en jaune les cellules non protégées de la plage A1:AZ256

Dim rCellule As Range

On Error GoTo Erreur:

For Each rCellule In [A1:AZ256]
    If Not rCellule.Locked Then
        rCellule.Interior.Color = vbYellow
    End If
    Application.StatusBar = "Traitement de la cellule " & rCellule.Address
Next

Exit Sub

Erreur:
    MsgBox "Erreur d'éxécution VBA " & Err.Description
End Sub
```

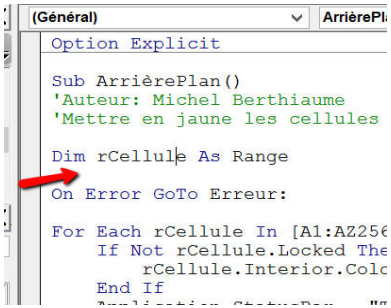
dans la zone de texte à droite:

Notez que
Option Explicit
ne devrait apparaître qu'une seule fois.

```
(Général)                                   ArrièrePlan
Option Explicit

Sub ArrièrePlan()
'Auteur: Michel Berthiaume
'Mettre en jaune les cellules non protégées de la plage A1:AZ256

Dim rCellule As Range

On Error GoTo Erreur:

For Each rCellule In [A1:AZ256]
    If Not rCellule.Locked Then
        rCellule.Interior.Color = vbYellow
    End If
    Application.StatusBar = "Traitement de la cellule " & rCellule.Address
Next

Exit Sub

Erreur:
    MsgBox "Erreur d'éxécution VBA " & Err.Description
End Sub
```

Testez votre code:

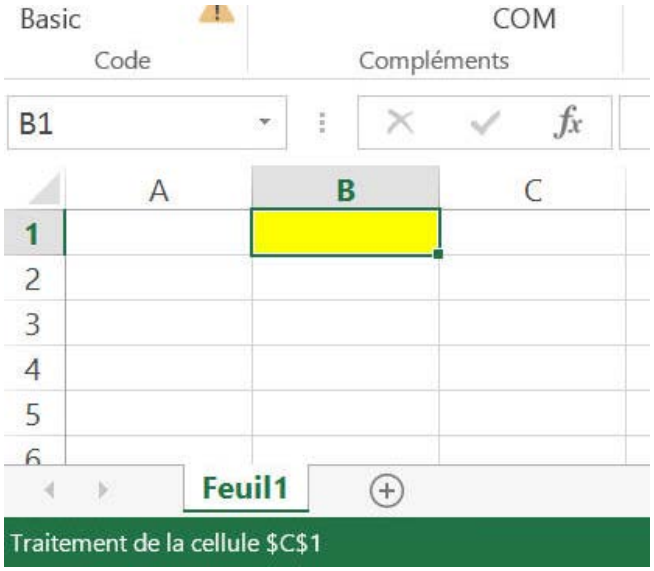
- Placez le curseur à l'intérieur de votre programme VBA (entre Sub et End Sub).
- Appuyez sur la touche F8.
- La ligne se colore en jaune




```
Sub ArrièrePlan()
'Auteur: Michel Berthiaume
'Mettre en jaune les cellules non protégées de la feuille active

Dim rCellule As Range

For Each rCellule In [A1:AZ256]
    If Not rCellule.Locked Then
        rCellule.Interior.Color = vbYellow
    End If
    Application.StatusBar = "Traitement de la cellule " & rCellule.Address
Next
End Sub
```

VBA exécute une ligne à chaque fois que vous enfoncez la touche F8, en répétant les instructions encadrées par For et Next pour chaque cellule de la feuille Excel active.

```
Sub ArrièrePlan()
'Auteur: Michel Berthiaume
'Mettre en jaune les cellules non protégées de la feuille active

Dim rCellule As Range

For Each rCellule In [A1:AZ256]
    If Not rCellule.Locked Then
        rCellule.Interior.Color = vbYellow
    End If
    Application.StatusBar = "Traitement de la cellule " & rCellule.Address
Next
End Sub
```



Vérifiez les changements dans la feuille Excel.

Remarquez que la barre d'état d'Excel indique l'adresse de la cellule traitée par VBA.

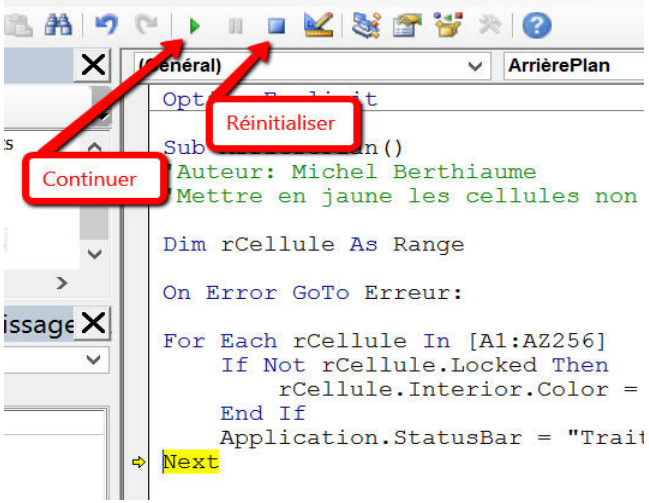
Pour terminer l'exécution du programme, cliquez sur les boutons

Continuer : le programme s'exécutera pour chaque cellule

- Si le sablier tarde à disparaître, arrêtez le programme en enfonçant ensembles les touches Ctrl et Pause Break

OU

Réinitialiser: le programme s'arrêtera.

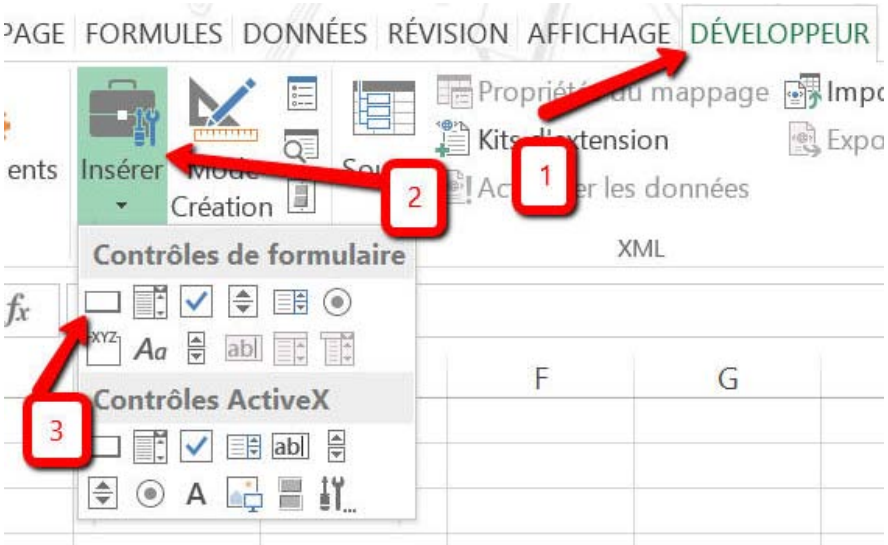


ATTENTION: vous devez terminer l'exécution d'un programme VBA selon la procédure ci-dessus

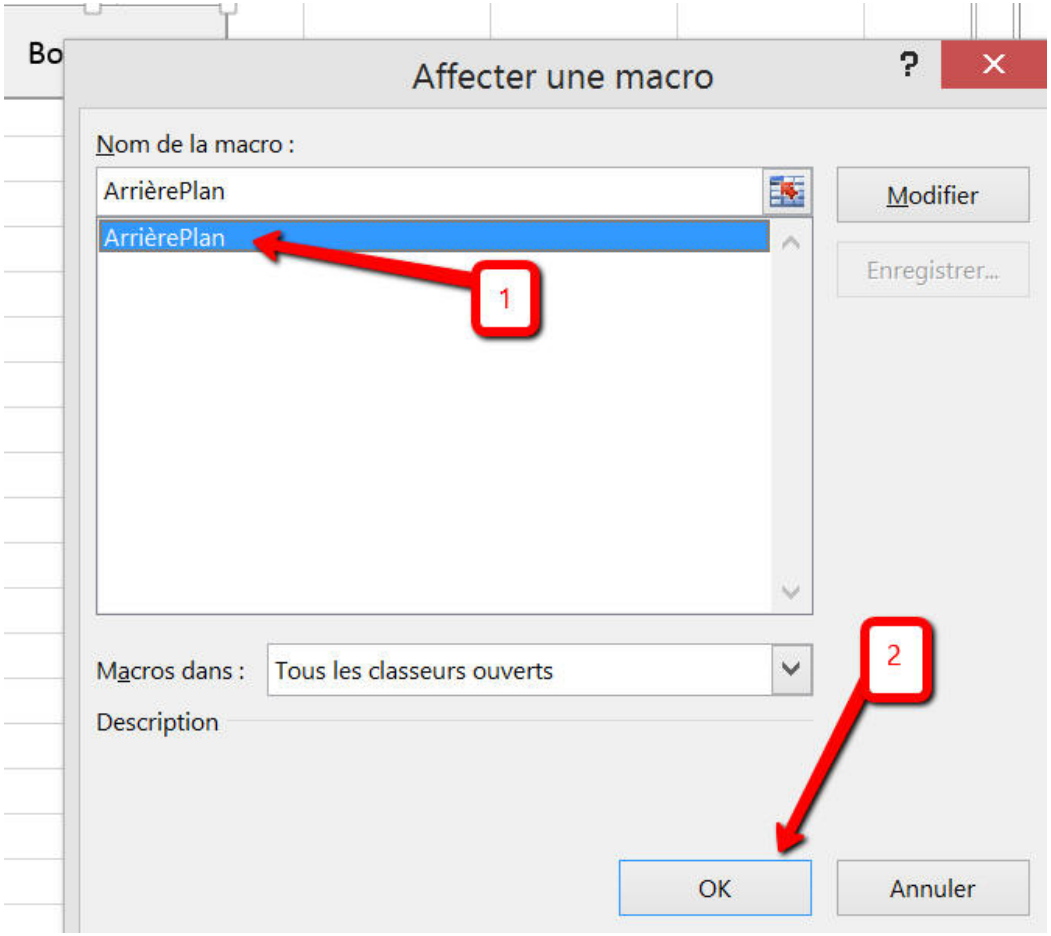
avant d'en exécuter un autre. En effet, si un programme est en débogage, VBA est en pause.

Rendez le programme disponible en Excel:

- Retournez dans la feuille Excel, onglet **développeur**.
- Dans le ruban **développeur**, bloc **Insérer Contrôle**,
  sélectionnez le Contrôle de formulaire **Bouton**.



- Dessinez un bouton dans la feuille active:
- Choisissez **ArrièrePlan** dans la liste.
- Cliquez Ok



Verrouillez quelques cellules jaunes, déverrouillez-en d'autres.
- Cliquez sur le bouton pour que les cellules déverrouillées se colorent en jaune.

Quelques commentaires sur ce programme:

- Il illustre l'un des 3 types de programme qu'on peut réaliser avec VBA: une procédure SUB.
- Il contient plusieurs catégories d'instructions:
  - Deux commentaires: 'Auteur: Michel Berthiaume et 'Mettre en jaune les cellules non protégées de la feuille active
  - Une déclaration: Dim rCellule As Range
  - Une boucle: For Each rCellule In [A1:AZ256] ... Next.
  - Un test: If ... Then ... End If.
  - Deux assignations: rCellule.Interior.Color = vbYellow et Application.StatusBar = "Traitement de la cellule " & rCellule.Address.
  - Une variable contenant un objet (une cellule Excel): rCellule.
  - Une constante VBA: vbYellow.
  - Un objet Excel (la barre d'état): Application.StatusBar.
  - Une référence à une plage Excel: [A1:AZ256].
  - Une constante littérale: "Traitement de la cellule ".
  - Deux opérateurs: Not et &.
  - Trois propriétés: Color (de l'objet rCellule.Interior), Locked et Address (de rCellule).
  - Une instruction de capture d'erreur: On Error GoTo
  - Une étiquette: Erreur:
- Il contient aussi une maladresse de programmation: une cellule jaune qui est verrouillée reste jaune après avoir été déverrouillée.

Si vous voulez enregistrer ce premier programme VBA Excel, suivez les recommandations dans le document

Enregistrer un classeur Excel contenant une macro VBA

Second exemple:
Fonction Excel qui compte le nombre de cellules d'une couleur donnée dans une plage donnée

Dans ce second exemple, on veut pouvoir écrire dans une cellule Excel la formule: **=fnNbCellulesCouleur(A1:D3;D1)** et ainsi afficher dans cette cellule le nombre de cellules de la plage A1:D3 dont la couleur est identique à la couleur de la cellule D1. Évidemment, on veut pouvoir remplacer A1:D3 et D1 par toute autre référence à une plage ou une cellule valide.

- Démarrez Excel avec un classeur vierge (ou utilisez le classeur créé pour l'exemple précédent).
- Colorez l'arrière-plan de quelques cellules.
- Démarrez l'éditeur VBA (voir l'exemple précédent).
- Créez un module ou utilisez un module existant.

Tapez (ou copiez-collez!) le code VBA:

```
Function fnNbCellulesCouleur(Plage As Range, Couleur As Range)
'Auteur: Michel Berthiaume
'Compter le nombre de cellules d'une couleur donnée dans une plage donnée
'Plage: plage de cellules à inspecter
'Couleur: cellule de la couleur cherchée

Dim rCellule As Range

For Each rCellule In Plage
    If rCellule.Interior.Color = Couleur.Interior.Color Then
        fnNbCellulesCouleur = fnNbCellulesCouleur + 1
    End If
Next
End Function

Sub Test()
Dim t
t = fnNbCellulesCouleur(Range("A1:D3"), Range("D1"))
End Sub
```

dans la zone de texte à droite:

```
Function fnNbCellulesCouleur(Plage As Range, Couleur As Range)
'Auteur: Michel Berthiaume
'Compter le nombre de cellules d'une couleur donnée dans une plage donnée
'Plage: plage de cellules à  inspecter
'Couleur: cellule de la couleur cherchée

Dim rCellule As Range

For Each rCellule In Plage
    If rCellule.Interior.Color = Couleur.Interior.Color Then
        fnNbCellulesCouleur = fnNbCellulesCouleur + 1
    End If
Next
End Function

Sub Test()
Dim t
t = fnNbCellulesCouleur(Range("A1:D3"), Range("D1"))
End Sub
```

VBA n'offre pas de moyen de tester directement une fonction. Procédons indirectement:

Placez le curseur sur la ligne **SUB TEST**.

```
        fnNbCellulesCoule
    End If
Next
End Function

Sub Test()
Dim t
t = fnNbCellulesCouleur(F
End Sub
```

Appuyez sur la touche F8.

La ligne se colore en jaune.

```
    End If
Next
End Function

Sub Test()
Dim t
t = fnNbCellulesCouleur(Range("A1
End Sub
```

VBA exécute une ligne à chaque fois que vous enfoncez la touche F8

D'abord la 3e ligne de **SUB TEST**.

```
End Function

Sub Test()
Dim t
t = fnNbCellulesCouleur(Range("A1:D3"), Range("
End Sub
```

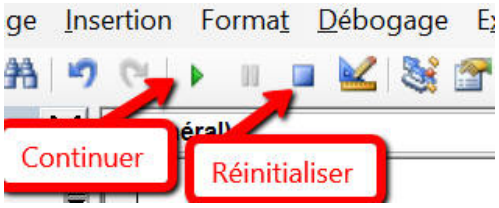
Puis en exécutant les lignes de la fonction.

```
End Sub

Function fnNbCellulesCouleur(Plage As Range, Couleur As Range)
'Auteur: Michel Berthiaume
'Compter le nombre de cellules d'une couleur donnée dans une plage donnée
'Plage: plage de cellules à  inspecter
'Couleur: cellule de la couleur cherchée

Dim rCellule As Range

For Each rCellule In Plage
    If rCellule.Interior.Color = Couleur.Interior.Color Then
        fnNbCellulesCouleur = fnNbCellulesCouleur + 1
    End If
Next
End Function

Sub Test()
Dim t
t = fnNbCellulesCouleur(Range("A1:D3"), Range("D1"))
End Sub
```

Appuyez sur la touche F8 pour exécuter chaque ligne de la fonction.

Pour terminer l'exécution de la fonction, cliquez sur les boutons

Continuer : la fonction s'exécutera pour chaque cellule



- Si le sablier tarde à disparaître, arrêtez le programme en enfonçant ensembles les touches Ctrl et Pause Break

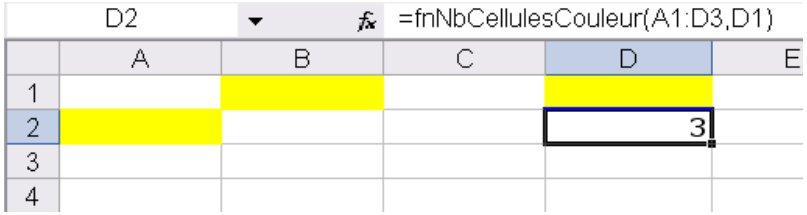
OU

Réinitialiser: le programme s'arrêtera.

ATTENTION: vous devez terminer l'exécution d'une fonction VBA selon la procédure ci-dessus avant d'exécuter un autre programme VBA. En effet, si un programme est en débogage, VBA est en pause.

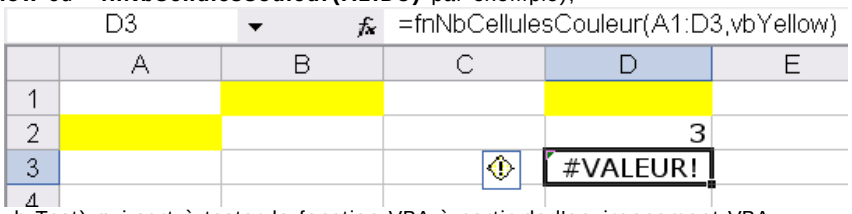
Utilisez la nouvelle fonction dans Excel.

- Dans une feuille du classeur Excel, colorez l'arrière plan de quelques cellules (A2, B1 et D1 par exemple).
- Dans une cellule de la feuille, inscrire la formule
  =fnNbCellulesCouleur(A1:D3;D1)
  en remplaçant **A1:D3** par la référence à une plage qui contient les cellules à dénombrer,
  et **D1** par la référence à une cellule de la couleur voulue.



Quelques commentaires sur le programme:

- Il illustre l'un des 3 types de programme qu'on peut réaliser avec VBA: une fonction personnalisée.
- Il contient plusieurs catégories d'instructions:
  - Plusieurs commentaires: les lignes commençant par '.
  - Deux paramètres de fonction: Plage et Couleur.
  - Quatre déclarations: fnNbCellulesCouleur(...), Plage As Range, Couleur As Range, rCellule As Range
  - Une boucle: For Each rCellule In Plage ... Next.
  - Un test: If ... Then ... End If.
  - Une expression logique: rCellule.Interior.Color = Couleur.Interior.Color
  - Une assignation: fnNbCellulesCouleur = fnNbCellulesCouleur + 1.
  - Une constante littérale: 1.
  - Un opérateur: +.
  - Une variable contenant un objet (une cellule Excel): rCellule.
  - Une propriété: Color.
- Il contient aussi une maladresse de programmation: si l'utilisateur entre des paramètres erronés
  (**=fnNbCellulesCouleur(A1:D3;vbYellow** ou **=fnNbCellulesCouleur(A1:D3)** par exemple),
  le message d'erreur n'est pas explicite.



- Il contient aussi une procédure Sub (Sub Test) qui sert à tester la fonction VBA à partir de l'environnement VBA.

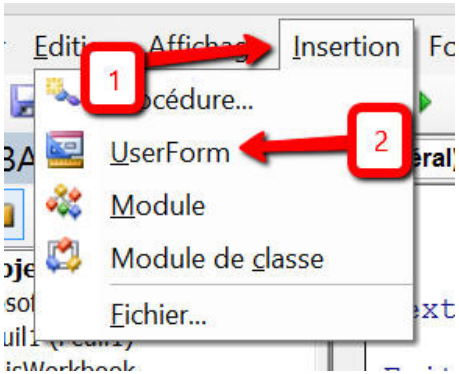
Troisième exemple:
Boîte de dialogue d'accueil

Dans ce troisième exemple, on veut qu'une fenêtre d'accueil s'affiche automatiquement lors de l'ouverture d'un classeur Excel.

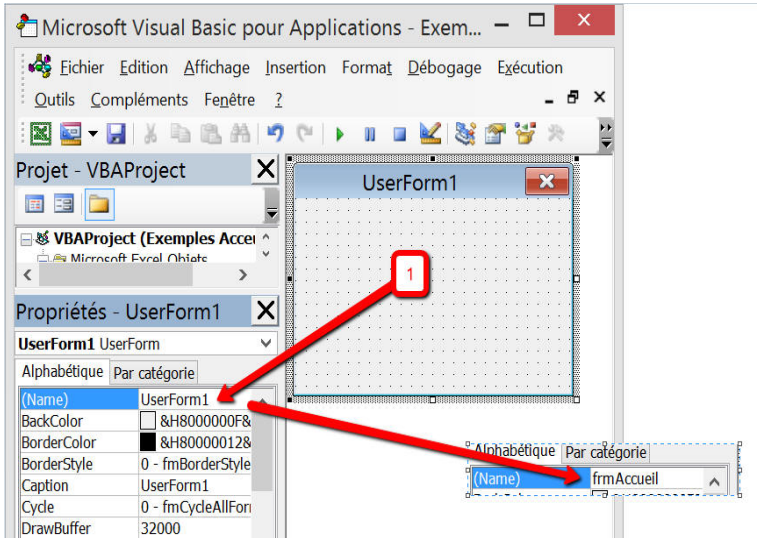
- Démarrez Excel et ouvrez un classeur (vierge ou non).
- Démarrez l'éditeur VBA (voir le premier exemple).

Créez un formulaire:

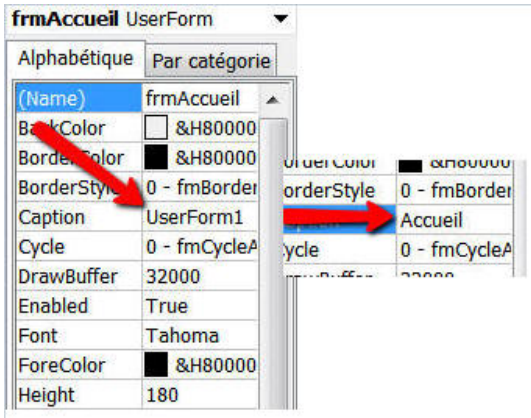
Du menu **Insertion**, sélectionnez **UserForm**:



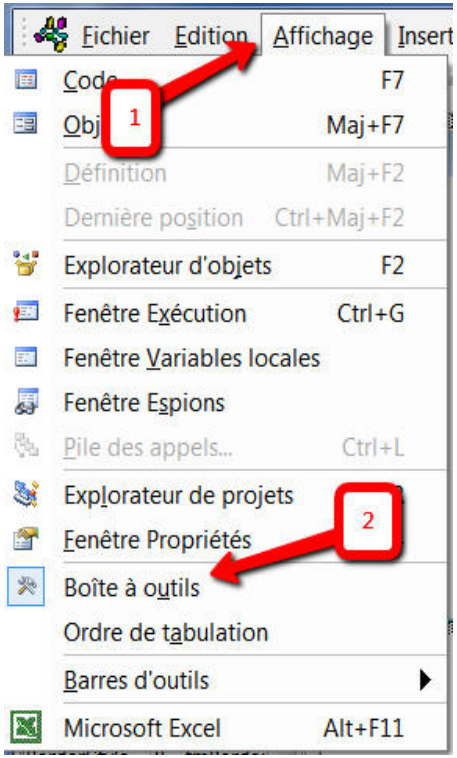
Changez le nom du formulaire dans la zone de texte (**Name**) en bas à gauche. Remplacez
UserForm1
par un nom significatif ne contenant pas d'espace
frmAccueil
par exemple.



Changez aussi le titre du formulaire, inscrit dans la zone de texte **Caption**, un peu plus bas

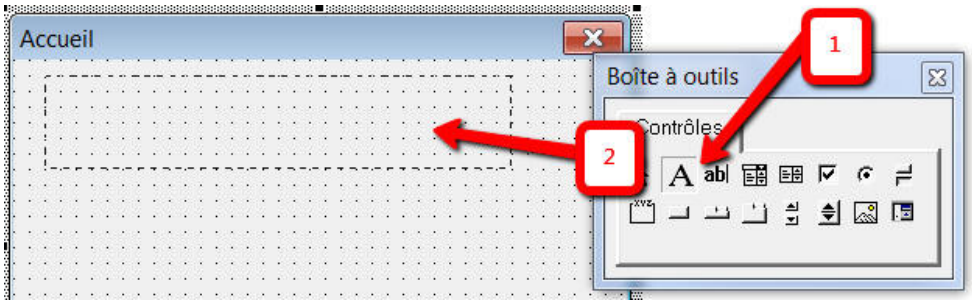


Du menu **Affichage**, sélectionnez Boîte à outils :

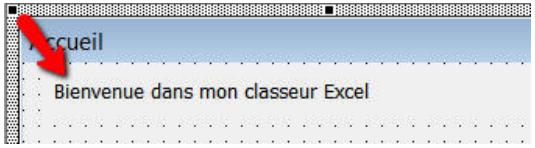


Dans la boîte à outils, sélectionnez un intitulé: A

Dessinez un libellé sur le formulaire.

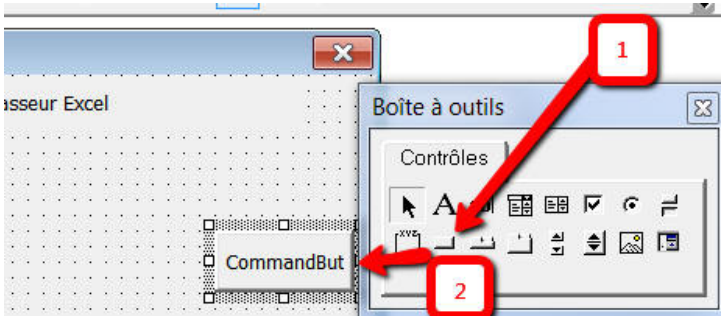


Cliquez dans le libellé et changez son contenu

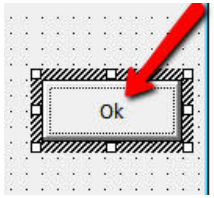


Dans la Boîte à outils, sélectionnez un bouton:

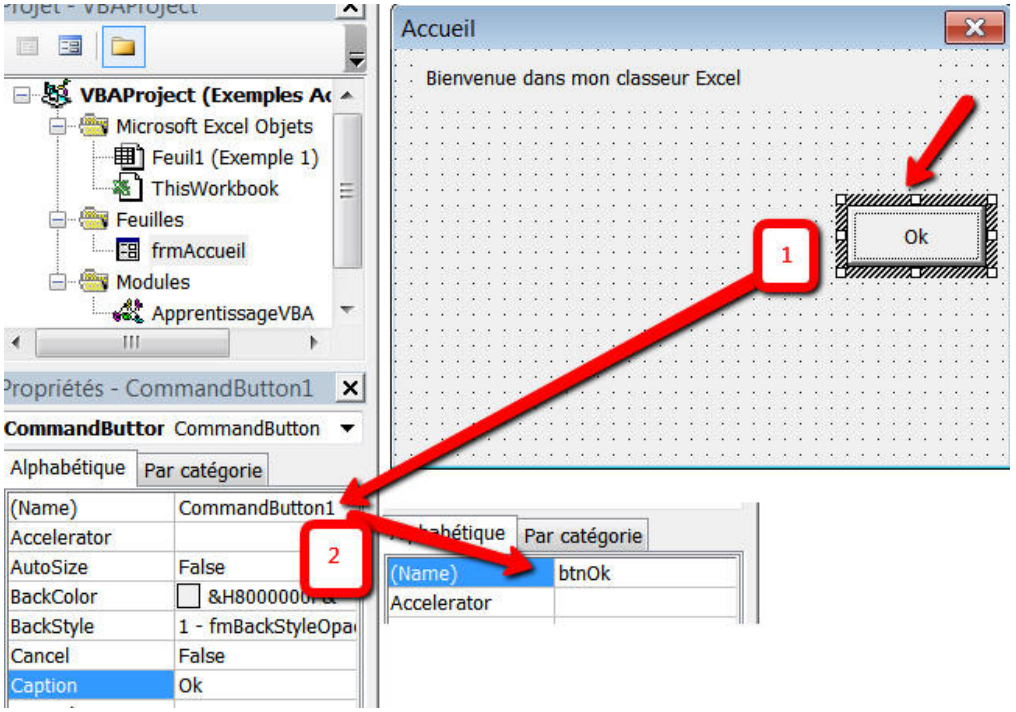
Dessinez un bouton sur le formulaire.



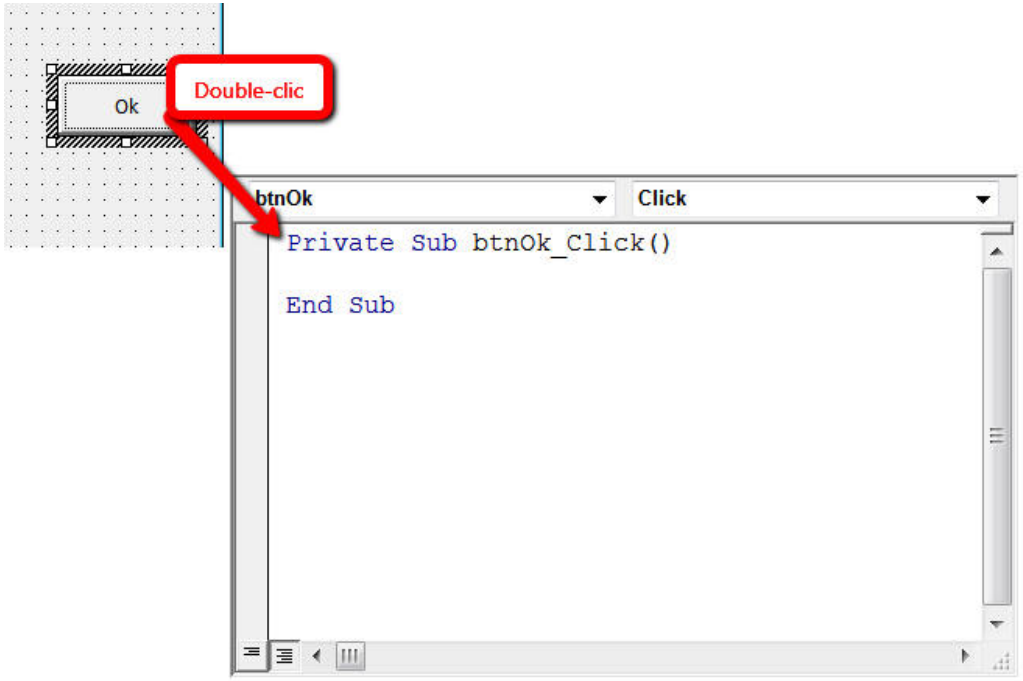
Cliquez dans le bouton et changez son libellé



Changez le nom du bouton dans la zone de texte (**Name**) en bas à gauche. Remplacez
CommandButton1
par
btnOk.



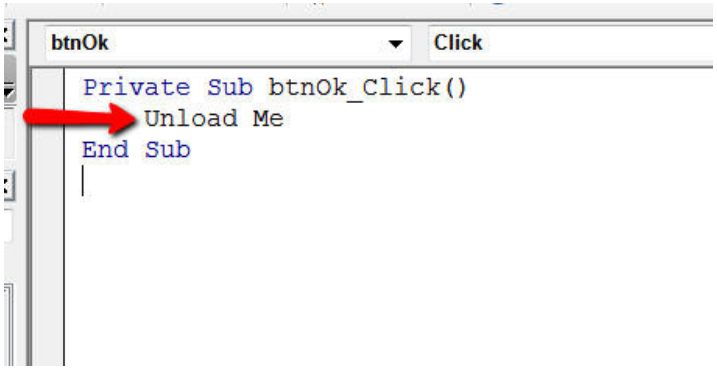
Double-cliquez sur le bouton **Ok**. Une fenêtre VBE s'ouvre



Tapez (ou copiez-collez!) le code VBA:

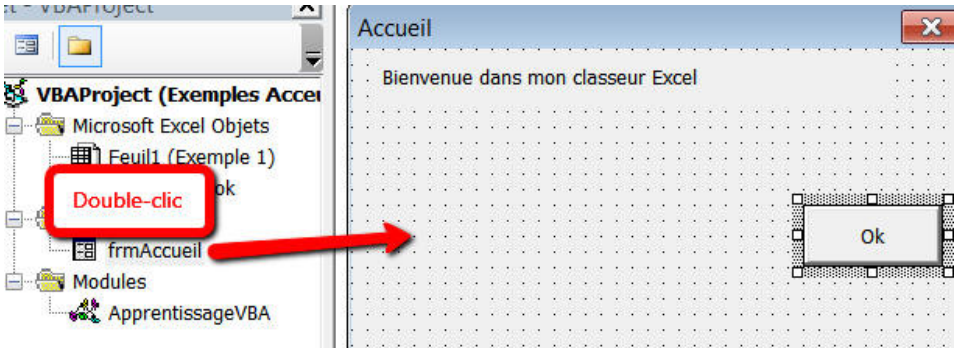
```
Unload Me
```

dans la zone de texte à droite:



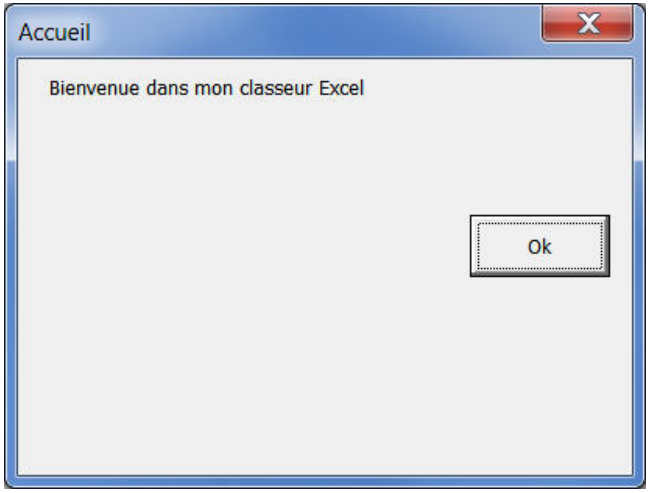
Testez le formulaire:

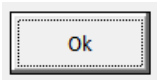
Double-cliquez sur **frmAccueil** pour vous assurer que le formulaire est affiché en mode création:



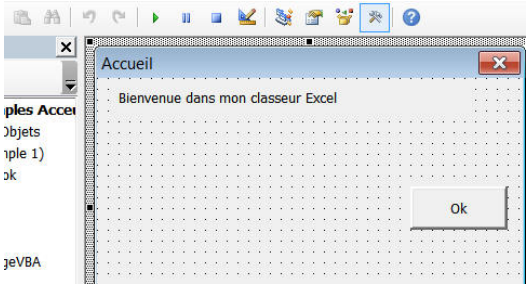
Appuyez sur la touche 

pour exécuter la boîte de dialogue.



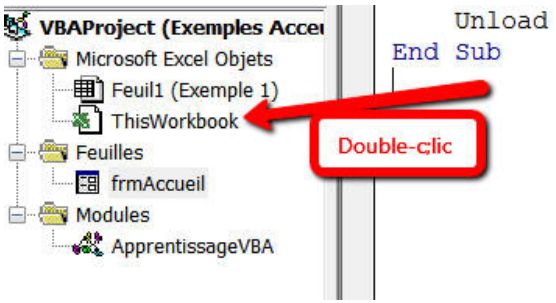
Appuyez sur le bouton 

pour la fermer.

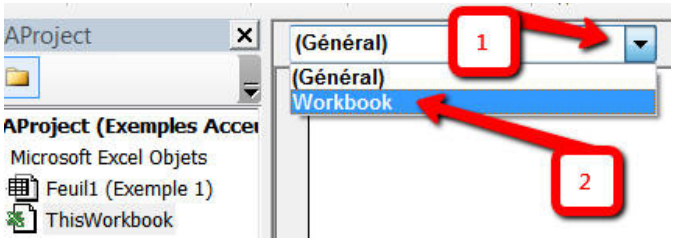


Associez l'ouverture du formulaire à l'ouverture du classeur Excel:

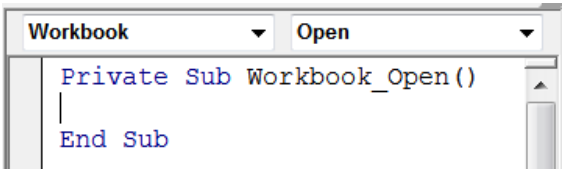
Dans l'explorateur de projet, double-cliquez sur **ThisWorkbook**



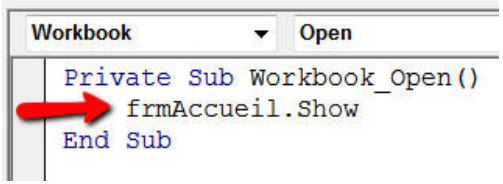
Dans la liste déroulante (général), choisir **Workbook**:

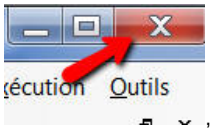


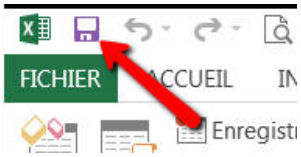
La seconde liste déroulante affiche **Open**



Dans la zone de texte, inscrire **frmAccueil.Show** entre les lignes **Private Sub** ... et **End Sub**:

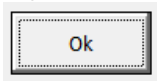


Fermez Visual Basic 

Sauvegardez le classeur Excel 

Et fermez-le

Ouvrez à nouveau le classeur. Il est probable que vous devrez activer les macros.
L'écran d'accueil s'affichera alors automatiquement.

Vous pouvez le fermer en appuyant sur 

Quelques commentaires sur le programme:

- Il illustre le dernier des 3 types de programme qu'on peut réaliser avec VBA: une boîte de dialogue (ou formulaire).
- Il contient des objets de formulaire
  - Un formulaire
  - Un libellé
  - Un bouton
- Il contient deux procédures événementielles:
  - Sub Workbook_Open() est exécutée lors de l'ouverture du classeur.
  - Sub btnOk_Click() est exécutée lorsque l'utilisateur appuie sur le bouton Ok
- Il contient quelques catégories d'instructions:
  - Une méthodes: Show de l'objet frmAccueil
  - Une instruction VBA : Unload
- Il contient aussi plusieurs maladresses de programmation, la plus grande étant qu'il ne s'efface pas de lui-même après quelques instants. De plus il n'est pas très joli.

Sommaire

Dans la présente section, nous avons vus 3 sortes de programmes VBA Excel

- Un programme normal: une procédure Sub qui identifie par une couleur les cellules déverrouillées dela feuille dee travail active
- Une fonction utilisateur qui compte le nombre de cellules d'une couleur donnée dans une plage donnée
- Une boîte de dialogue d'accueil

Nous avons aussi présenté plusieurs modes d'emploi en format pdf:


Afficher l'onglet DÉVELOPPEUR


Activer l'exécution des macros


Enregistrer un classeur Excel contenant une macro VBA


Déverrouiller des cellules Excel


Enregistrer un classeur Excel contenant une macro VBA

Fichier exemple

Le fichier Excel Tutoriel Exemples Accueil.xlsm contient les exemples présentés dans la présente section.

But et limites du tutoriel

Ce tutoriel est destiné à des étudiants de niveau universitaire en administration, n'ayant pas de connaissance en programmation, mais maîtrisant Excel.
Certains aspects de VBA ont été volontairement omis parce que jugés inutiles à cet auditoire.
Pour une description complète du langage, consultez un bon livre!

Références

VBA:

http://msdn.microsoft.com/fr-fr/library/office/fp179694(v=office.15).aspx Mise en route avec VBA dans Excel 2013

http://ericrenaud.fr/ Astuces et code VBA pour Excel

http://www.vbfrance.com/ Visual Basic Codes Sources. Les pages sont longues à charger.

Excel:

http://www.cathyastuce.com Cathy Astuces: astuces, cours, exercices sur Office et Windows.
http://www.admexcel.com Excel par l'exemple
http://www.misfu.com/cours-et-tutoriaux-tableur-excel.html Cours / Formation Excel
http://www.excel-online.net Le site pour maitriser le tableur Excel

Livres:

VB & VBA in a Nutshell: The Language, par Paul Lomax: Un des meilleurs livres de référence VBA. Pour programmeurs.

Suite: Concepts de programmation

VBA Excel

par Michel Berthiaume

http://www.usherbrooke.ca/adm/departements/simqg/professeurs/michel-berthiaume/

Concepts de programmation

Sur cette page...

Les prochaines pages...

Qualités d'un bon programme

Un programme informatique est un peu comme une recette de cuisine. Il requiert des ingrédients (données, variables, objets), contient une série de manipulations à faire avec ces ingrédients (instructions VBA) en utilisant des outils plus ou moins performants (VBE, Excel, Windows) et donne un résultat quelquefois plus intéressant que les ingrédients pris séparément.

Il y a de bonnes recettes, et de moins bonnes recettes. Il en est de même pour les programmes.

Un bon programme:

- Fait **toujours** ce qui est prévu.
- Ne fait **jamais** ce qui n'est pas prévu.
- Est facile à **utiliser**.
- Est facile (peu coûteux) à **modifier**.

Très peu de programmes disponibles sur le marché actuel répondent à ces critères.
Lorsque vous aurez terminé l'écriture d'un programme, demandez-vous s'il répond aux quatre critères de façon satisfaisante pour l'usage prévu.

Une suggestion dès le départ: gardez vos programmes simples.

Types de programme

Du point de vue de l'utilisation (et donc de la programmation), un programme **VBA** Excel peut être:

- L'automatisation d'une série d'opérations qu'on peut faire manuellement dans Excel: la procédure **SUB**.
- Une fonction Excel que les programmeurs Microsoft n'ont pas prévue: la procédure **Function** (fonction personnalisée).
- Une boîte de dialogue qui permet des échanges avec l'utilisateur autrement que par les moyens prévus par les programmeurs Microsoft: le formulaire.

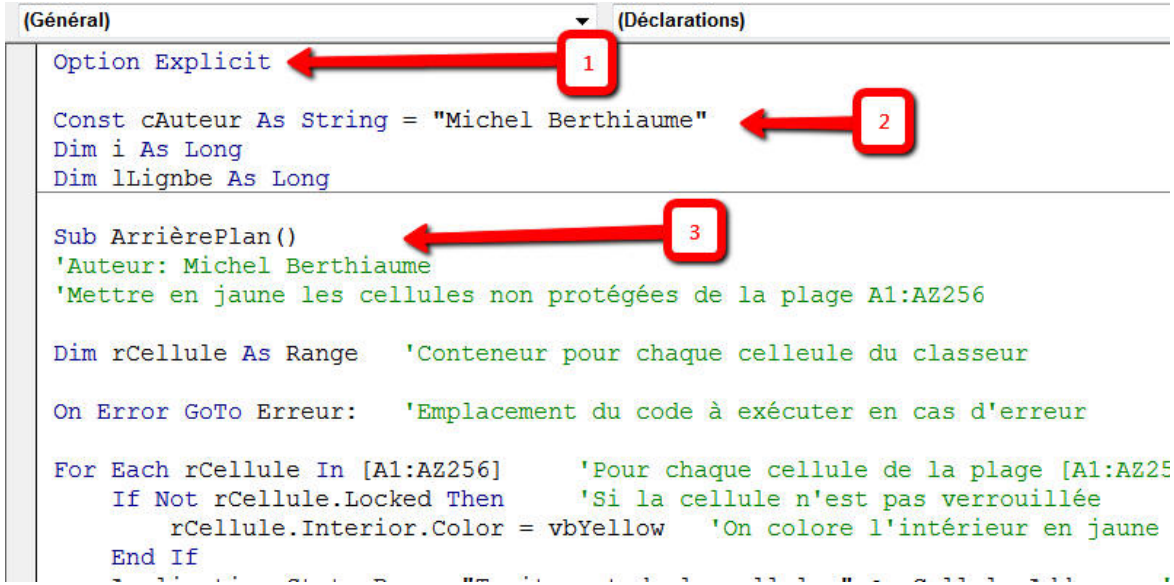
- La création d'une classe.

On trouve un exemple de chacun (sauf la création de classe) dans la page Introduction du présent tutoriel.

Composition d'un module VBA

Une module VBA se compose des trois éléments suivants:

1. L'instruction **Option Explicit**, qui force la déclaration explicite des variables.
2. Des déclarations de variables de niveau module.
3. Une ou des procédures (programmes):



Une procédure doit toujours être encadrée par les instructions **Sub...End Sub** ou **Function ... End Function**.

[**Private** | **Public**] [**Static**] **Sub** *nom* [*(liste de paramètres)*]
[instructions]
[**Exit Sub**]
[instructions]
End Sub

Où:

- **Public** Indique que la procédure **Sub** est accessible à toutes les autres procédures dans l'ensemble des modules. Valeur par défaut.
- **Private** Indique que la procédure **Sub** n'est accessible qu'aux autres procédures du module dans lequel elle a été déclarée.
- **Static** Indique que les valeurs des variables locales de la procédure Sub sont conservées entre les appels.En l'absence de **Static**, les variables sont réinitialisées à chaque exécution de la procédure. **Static** est très rarement nécessaire.
- *nom* Nom de la procédure Sub. Doit especter les règles des noms de variables.
- *Liste de paramètres* Liste de variables représentant des paramètres qui sont passés à la procédure Sub lorsqu'elle est appelée. Les variables multiples sont séparées par des virgules. Chaque paramètre doit être déclaré dans la forme suivante:
  [**Optional**] [**ByVal** | **ByRef**] [ParamArray] *nom*[( )] [As **type**] [=*défaut*]
  où:
  - **Optional** indique que ce paramètre n'est pas obligatoire. S'il est utilisé, tous les paramètres suivants doivent l'être aussi.
  - **ByRef** indique que si la valeur du paramètre est modifiée dans la procédure, elle le sera aussi dans la procédure appelante. **ATTENTION,** en VBA **ByrRef** est la valeur par défaut, au contraire de la plupart des langages de programmation.
  - **ByVal** indique que si la valeur du paramètre est modifiée dans la procédure, cela n'affecte pas sa valeur dans la procédure appelante.
  - **ParamArray** Indique que le paramètre suivant est un tableau. **ParamArray** ne peut précéder que le dernier paramètre de la liste.
  - *Nom* nom du paramètre, doit respecter les règles des noms de variables.
  - **type** type du paramètre.
  - *défaut* constante initialisant un paramètre optionnel (**Optional**)

- *instructions* Tout groupe d'instructions à exécuter dans la procédure **Sub**.
- **Exit Sub** Instruction VBA permettant de terminer l'exécution de la procédure avant la fin.
- **End Sub** Instruction délimitant la fin de la procédure **Sub**.

[**Private** | **Public**] [**Static**] **Function** *nom* [*(liste de paramètres)*] [**AS** *type*]
[instructions]
[*nom* = *expression*]
[**Exit Function**]
[instructions]
[*nom* = *expression*]
End Function

Où:

- **Public** Indique que la procédure **Function** est accessible à toutes les autres procédures dans l'ensemble des modules.
- **Private** Indique que la procédure **Function** n'est accessible qu'à d'autres procédures du module dans lequel elle a été déclarée.
- **Static** Indique que les valeurs des variables locales de la procédure Sub sont conservées entre les appels.En l'absence de **Static**, les variables sont réinitialisées à chaque exécution de la procédure. **Static** est très rarement nécessaire.
- *nom* Nom de la procédure Function. Respecte les règles des noms de variables.
- *Liste de paramètres* Liste de variables représentant des paramètres qui sont passés à la procédure Sub lorsqu'elle est appelée. Les variables multiples sont séparées par des virgules. Chaque paramètre doit être déclaré dans la forme suivante:
  [**Optional**] [**ByVal** | **ByRef**] [ParamArray] *nom*[( )] [As **type**] [=*défaut*]
  où:
  - **Optional** indique que ce paramètre n'est pas obligatoire. S'il est utilisé, tous les paramètres suivants doivent l'être aussi.
  - **ByRef** indique que si la valeur du paramètre est modifiée dans la procédure, elle le sera aussi dans la procédure appelante. **ATTENTION,** en VBA **ByRef** est la valeur par défaut, au contraire de la plupart des langages de programmation.
  - **ByVal** indique que si la valeur du paramètre est modifiée dans la procédure, cela n'affecte pas sa valeur dans la procédure appelante.
  - **ParamArray** Indique que le paramètre suivant est un tableau. **ParamArray** ne peut précéder que le dernier paramètre de la liste.
  - *Nom* nom du paramètre, respectant les règles des noms de variables.
  - **type** type du paramètre.
  - *défaut* constante initialisant un paramètre optionnel (**Optional**)
- *Type* Type de la valeur retournée par la procédure Function.
- *instructions* Tout groupe d'instructions à exécuter dans la procédure **Function**
- *expression* Valeur retournée par la procédure **Function**.
- **Exit Function** Instruction VBA permettant de terminer l'exécution de la procédure avant la fin.
- **End Function** Instruction délimitant la fin de la procédure **Function**.

Sub ou Function?

Une procédure **Sub** sans paramètre obligatoire:

- Ne retourne pas de valeur.
- Peut modifier le contenu de la feuille Excel.
- Peut être associée à un objet (bouton, graphique) dans la feuille Excel et exécutée lorsque l'objet est cliqué.

Une procédure **Sub** avec paramètre obligatoire:

- Ne retourne pas de valeur.
- Peut être exécutée lorsqu'un événement Excel se produit (ouverture, fermeture, modification de cellule...).

Une procédure **Function**:

- Retourne une valeur.
- Ne doit pas modifier la feuille de travail.
- Ne doit pas afficher de boîte de dialogue.
- Ne peut être utilisée que dans une formule Excel ou une procédure VBA.

Composantes d'une procédure

Une procédure VBA comporte un nom, de la documentation, des objets, des instructions et une gestion des erreurs:

- Un nom: Assignez toujours un nom qui décrit la nature de votre procédure. Ça la rend plus facile à **modifier.**
- e

De la documentation: au minimum, indiquez dès la 2 ligne le nom de l'auteur et dès la 3e ligne le but de la procédure. N'hésitez pas à la parsemer de commentaires qui vous permettront de mieux comprendre ce qu'elle fait (et comment elle le fait) quand vous voudrez **la modifier** dans un ou dix ans.

- Des objets: une recette contient des ingrédients, une procédure contient des objets. Ces objets peuvent être:
  - Des cellules Excel (leur contenu. leur couleur, leur format, ...).
  - D'autres objets apparaissant à l'écran (graphique, feuille, onglet, imprimante, ...).
  - Des valeurs variables (compteurs, accumulateurs, ...).
  - Des valeurs constantes (Pi, code de couleur, ...).
- Des instructions:
  - Assignation, conversion.
  - Action.
  - Contrôle.
- Une gestion des erreurs: des instructions à exécuter lors que l'inattendu se produit.

Une boîte de dialogue contient des contrôles, nom qu'on donne aux objets de formulaire qui apparaissent à l'écran (boutons, zones de texte, cases, ...).

Pour développer une procédure, on utilise un environnement de développement. Cet environnement est le même pour tous les programmes de la suite Office: VBE (Visual Basic Editor).

Le code d'une procédure VBA est enregistrée dans un classeur, dans une feuille, dans un module ou dans un formulaire. Le tout est enregistré à l'intérieur du document Office. Depuis Office 2007, les noms des documents contenant du code VBA doivent avoir une extension spécifique (.xlsm au lieu de .xlsx).

Pour **sauvegarder** un programme VBA, il faut sauvegarder le **fichier Excel** qui le contient.

Voir le document

Enregistrer un classeur Excel contenant une macro VBA

Documentation

On documente une procédure VBA à l'aide de commentaires. Chaque commentaire doit être précédé d'une apostrophe. Tout ce qui est à droite d'une apostrophe sera ignoré lors de l'exécution de la procédure.

S'il y a du code VBA à gauche de la première apostrophe d'une ligne, il sera exécuté.

VBE rend en vert les parties identifiées comme commentaires.

Ci-dessous un exemple de procédure Sub documentée:

```
Sub ArrièrePlan()
'Auteur: Michel Berthiaume
'Mettre en jaune les cellules non protégées de la feuille active

Dim rCellule As Range

On Error GoTo Erreur:   'Active la gestion des erreur d'exécution

    For Each rCellule In [A1:AZ256]     'Pour chaque cellule de la plage [A1:AZ256]
        If Not rCellule.Locked Then     'Si la cellule n'est pas verrouillée
            rCellule.Interior.Color = vbYellow  'On colore l'intérieur en jaune
        End If
        Application.StatusBar = "Traitement de la cellule " & rCellule.Address  'Affiche l'adresse de la cellule traitée
    Next    'On passe à la cellule suivante

Exit Sub    'Fin du trairement normal

Erreur:     'Instructions à exécuter en cas d'erreur

    MsgBox Err.Description  'Affiche le message spécifique à l'erreur

End Sub
```

La présentation du code contribue aussi à la documentation.

Remarquez l'utilisation de lignes vides et de tabulations pour marquer les différents blocs de code.

L'instruction For est alignées avec le Next correspondant et le contenu de la boucle est décalé.
Il en est de même avec l'instruction If et son End If. Le contenu est aussi décalé.

Sommaire

Dans cette section, nous avons vu

- Ce qui permet de mesurer la qualité d'un programme VBA
- La différence entre Sub et Function,
- Les composantes d'un programme VBA
- Comment documenter un programme VBA

Suite: VBE: l'éditeur VBA

VBA Excel

par Michel Berthiaume

http://www.usherbrooke.ca/adm/departements/simqg/professeurs/michel-berthiaume/

VBE: l'éditeur VBA

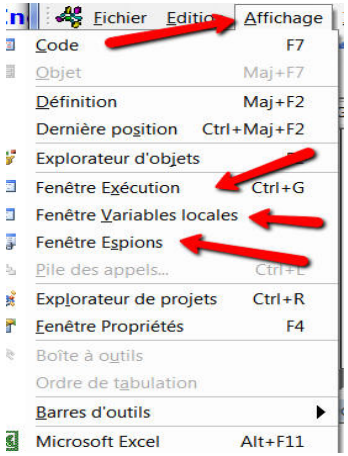
Sur cette page...

Les prochaines pages...

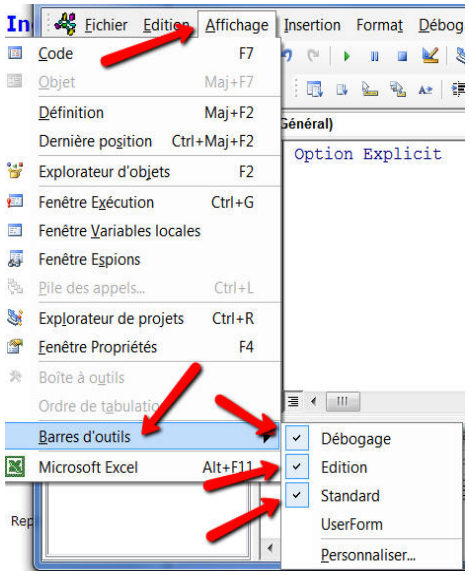
Vue d'ensemble

Avant d'explorer les composantes de l'éditeur VBE, vous devez l'afficher:

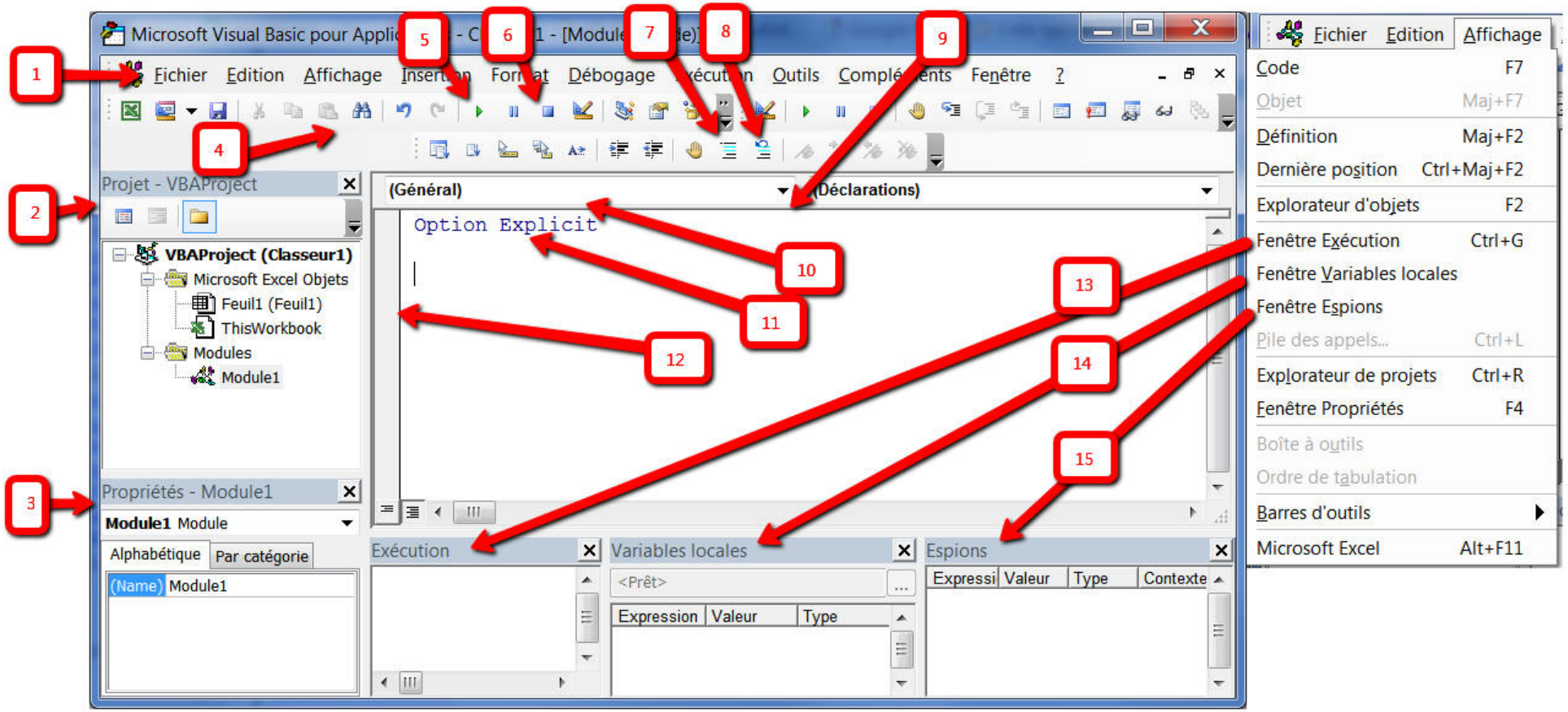
De plus, assurez-vous d'afficher les fenêtres optionnelles:



Finalement, affichez les barres d'outils optionnelles:



L'environnement de développement VBE comporte plusieurs zones:



Le tableau ci-dessous explique les éléments qu'on retrouve à l'écran.
Utilisez le menu **Affichage** pour afficher les éléments absents de votre affichage.

1) Menus	Les choix les plus fréquemment utilisés sont dans les menus Insertion et Débogage.
2) Explorateur de projets	Sert à choisir ce qui est affiché dans la fenêtre de code (9) et dans l'explorateur d'objets (3).
3) Explorateur d'objets	Permet de consulter et modifier les propriétés de l'objet sélectionné dans l'explorateur de projets (2).
4) Barres d'outils	Il est recommandé d'afficher toutes les barres d'outils optionnelles.
5) Bouton Démarrer/Continuer	Sert à démarrer l'exécution continue du code sélectionné dans la fenêtre de code (9).
6) Bouton Arrêter	Sert à arrêter l'exécution du code VBA en cours.
7) Bouton commenter	Sert à désactiver (mettre en commentaire) le bloc de code sélectionné dans la fenêtre Code (9).
8) Bouton ne pas commenter	Sert à réactiver (enlever commentaire) le bloc de code sélectionné dans la fenêtre Code.
9) Fenêtre Code OU Fenêtre Formulaire	Affiche le code VBA **OU** les objets de formulaire de l'objet sélectionné dans l'explorateur de projets (2).
10) Nom du module affiché	Liste déroulante permettant de passer d'un objet du projet à un autre sans passer par l'explorateur de projets (2).
11) Option Explicit	Instruction que l'on doit retrouver au début de chaque module et qui force la déclaration explicite des variables.
12) Marge de la fenêtre Code	Contient des repères visuels lors du développement d'un programme VBA.
13) Fenêtre Exécution	
14) Fenêtre Variables locales	
15) Fenêtre Espions	

Modifications essentielles à l'interface

- L'instruction **Option Explicit** peut et doit apparaître au début de chaque module VBA. Vous devez automatiser son apparition.

- Les boutons Commenter bloc et Ne pas commenter bloc vous seront utiles pour rendre inopérantes/opérantes des instructions erronées ou suspectes lors de tests et débogages.
  Ils font partie de la barre de boutons débogage.
  Vous pouvez aussi les ajouter à la barre d'accès rapide.

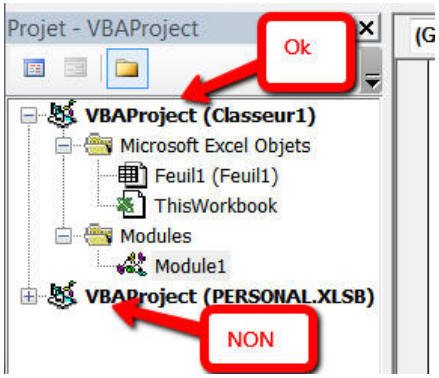
Pour modifier l'interface VBE selon ces conseils:

Modifier l'interface VBE

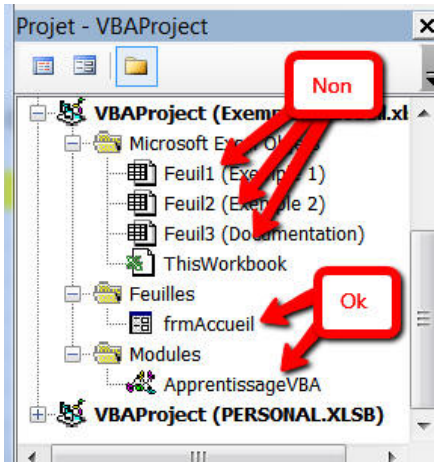
L'explorateur de projets (sauvegardes)

L'explorateur de projets permet de naviguer dans les différents endroits pouvant contenir du code VBA Excel, c'est à dire les différents emplacements où il vous est possible de sauvegarder le code que vous allez écrire.

Vos choix:



- Un classeur Excel: le code VBA est sauvegardé avec le classeur. Si vous copiez le classeur, il est copié avec. Si vous supprimez le classeur, il est supprimé.
- Le classeur de macros personnelles: le code VBA est sauvegardé dans le classeur personal.xlsb de l'ordinateur, **ET NON DANS LE CLASSEUR ACTIF**. Comme personal.xlsb est automatiquement chargé lors du démarrage d'Excel sur un ordinateur, les programmes VBA sauvegardés dans le classeur personal.xlsb de cet ordinateur sont disponibles dans tous les classeurs ouverts sur ce même ordinateur. Ce choix est utile aux programmeurs expérimentés et aux administrateurs de parcs d'ordinateurs. **Il est fortement déconseillé aux programmeurs amateurs.**



Dans un classeur Excel, vous devez choisir un des endroits suivants:

- Une feuille de ce classeur. Cet emplacement déconseillé.
- Le classeur lui-même (**ThisWorkbook**). Il est utilisé pour enregistrer les programmes déclenchés par des événements de classeur (ouverture, fermeture, modification de cellule...).
- Un module. Il est utilisé pour enregistrer les programmes utilisés dans le classeur ou par les autres programmes du classeur.
  - Un formulaire. Il est utilisé pour enregistrer les programmes déclenchés par les événements du formulaire.

Tous ces emplacements peuvent contenir du code VBA Excel.

En pratique, vous utiliserez le plus souvent les endroits suivants:

- Un module dans un classeur Excel.
- Un formulaire dans un classeur Excel.

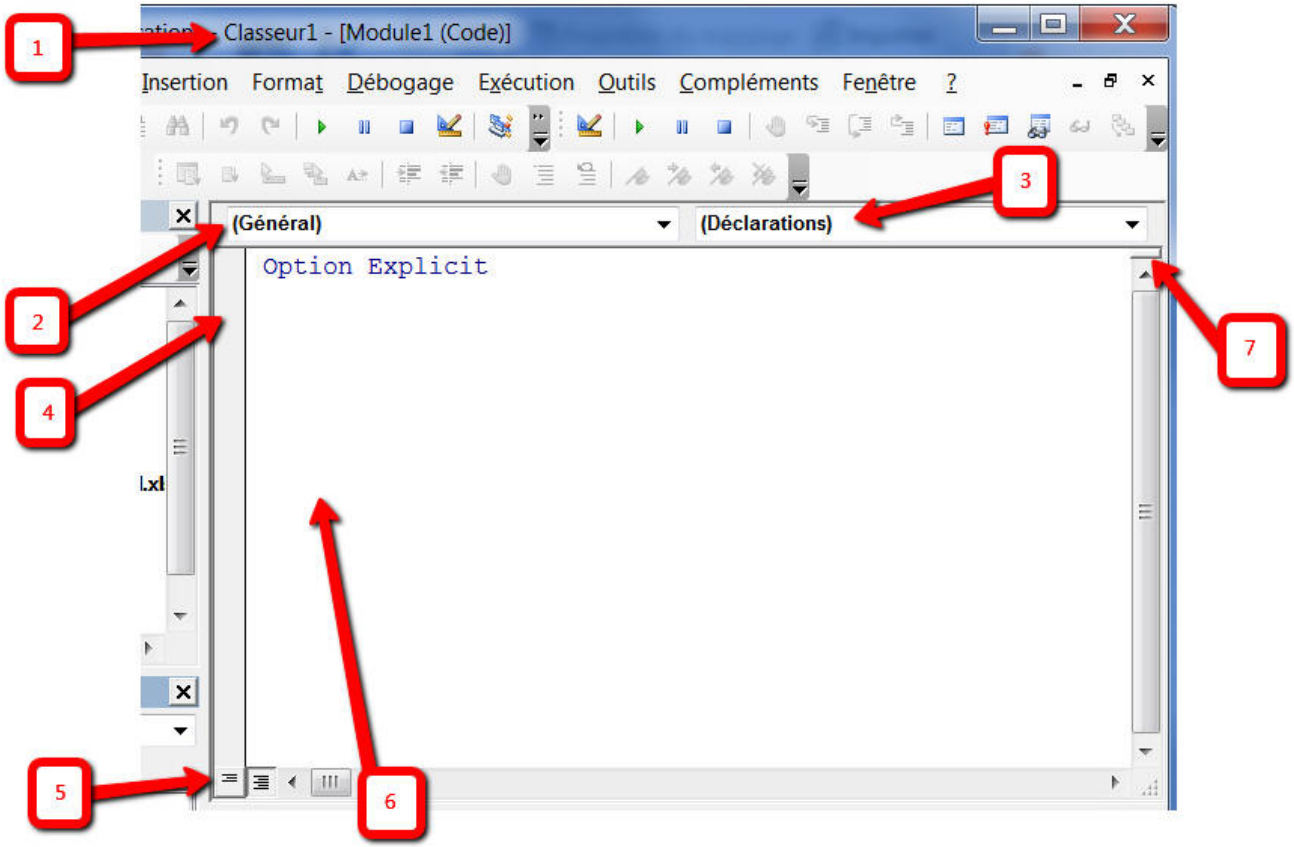
Explorateur d'objets (fenêtre Propriétés)

L'explorateur d'objets sert principalement:

- À renommer les modules et formulaires.
- À modifier les propriétés des contrôles (objets) de formulaires

Fenêtre Code ou fenêtre Formulaire

C'est dans cette fenêtre qu'on crée/édite le code VBA.



1) Entête	Affiche le nom de l'emplacement (classeur, feuille de classeur, module, formulaire) contenant le code affiché. L'emplacement est choisi dans l'explorateur de projets.
2) Zone objet	Objets disponibles dans le contexte du conteneur choisi dans l'explorateur de projets.
3) Zone procédures/ événements	Liste de toutes les procédures du module (en gras) et des procédures événementielles (en pâle) possibles de l'objet choisi dans la zone objet (2).
4) Marge de la fenêtre code	Contient des repères visuels lors du développement d'un programme VBA: les points d'arrêt.
5) Affichage Procédure/module	Boutons permettant de basculer entre l'affichage d'une seule procédure et l'affichage de toutes les procédures du module.
6) Éditeur de code	Éditeur de texte permettant de créer/éditer des procédures SUB ou des procédures Function.
7) Barre de fractionnement	Permet de diviser la fenêtre de code en deux parties horizontales pour afficher simultanément deux parties d'un même module.

La fenêtre code offre les fonctionnalités d'éditeur habituelles pour créer/éditer du code VBA. Il y existe aussi quelques fonctionnalités additionnelles:

- Les touches **Ctrl+espace** offrent de compléter le mot où se trouve le curseur dans la fenêtre de code.
- Les mots réservés VBA sont bleus.
- Les commentaires sont verts.
- Les lignes erronées sont rouges.
- La touche **F1** démarre l'aide en ligne, dont le principal défaut est l'absence d'outil de recherche.

D'autres raccourcis sont disponibles.

Fenêtre Exécution

Cette fenêtre permet d'exécuter une ligne de code VBA en dehors d'un module.

Par exemple, taper

```
? fnMoyenneAvecCellulesNulles(Range("A1..B5")
```

exécute la fonction personnalisée fnMoyenneAvecCellulesNulles et en affiche le résultat.

```
? Range("A1")
```

affichera la valeur de la cellule A1.

Modérément utile pour tester des instructions. Si vous n'utilisez pas cette fonctionnalité, vous devriez effacer cette fenêtre.

Fenêtre variables locales

Cette fenêtre affiche la liste des variables locales du module en cours d'exécution (mode débogage) et permet d'en connaître la valeur et le type.

Cette fenêtre est très utile pour déboguer les programmes.

Fenêtre Espions

Cette fenêtre affiche la valeur de variables choisies, en mode débogage. À la différence de la Fenêtre variables locales, il est possible de s'y faire afficher la valeur de n'importe quelle variable, locale ou non.

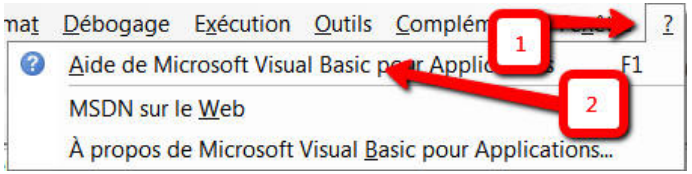
Modérément utile pour déboguer les programmes. Si vous pourriez effacer cette fenêtre.

Aide VBA en ligne

Dans Office 2013, il n'est plus possible d'accéder à l'aide locale.

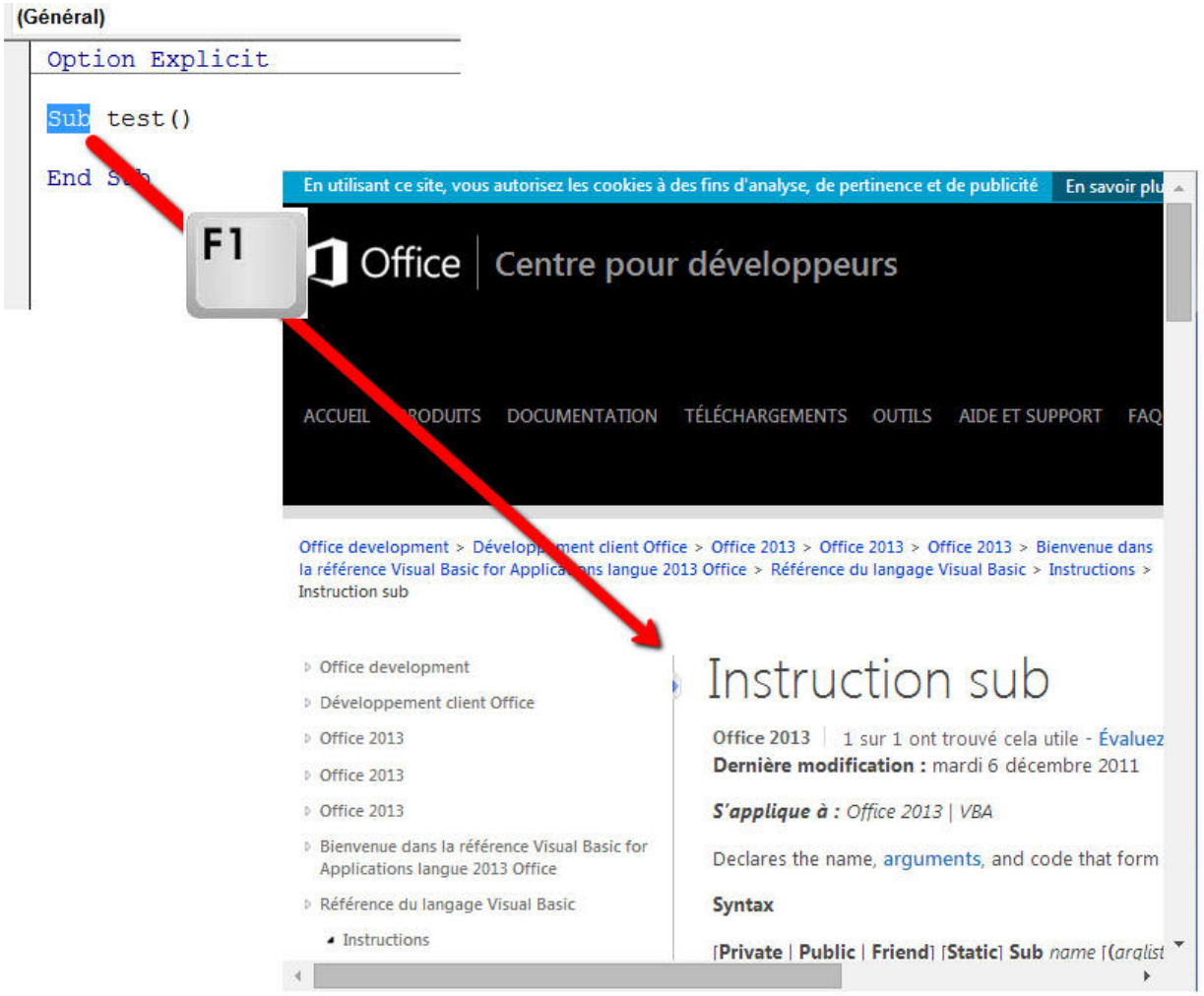
On peut accéder à l'aide VBA en ligne de 2 façons:

- Cliquez ?, puis Aide de Microsoft Visual Basic pour Applications:



- Enfoncez la touche 

:



L'aide VBA tente d'identifier le sujet sur lequel vous souhaitez de l'aide et d'afficher l'aide correspondante.

Il est aussi possible de consulter la référence de Microsoft sur le langage VBA lui-même, à l'adresse:
http://msdn.microsoft.com/en-us/library/jj692818(v=office.15).aspx

Aide VBA Locale

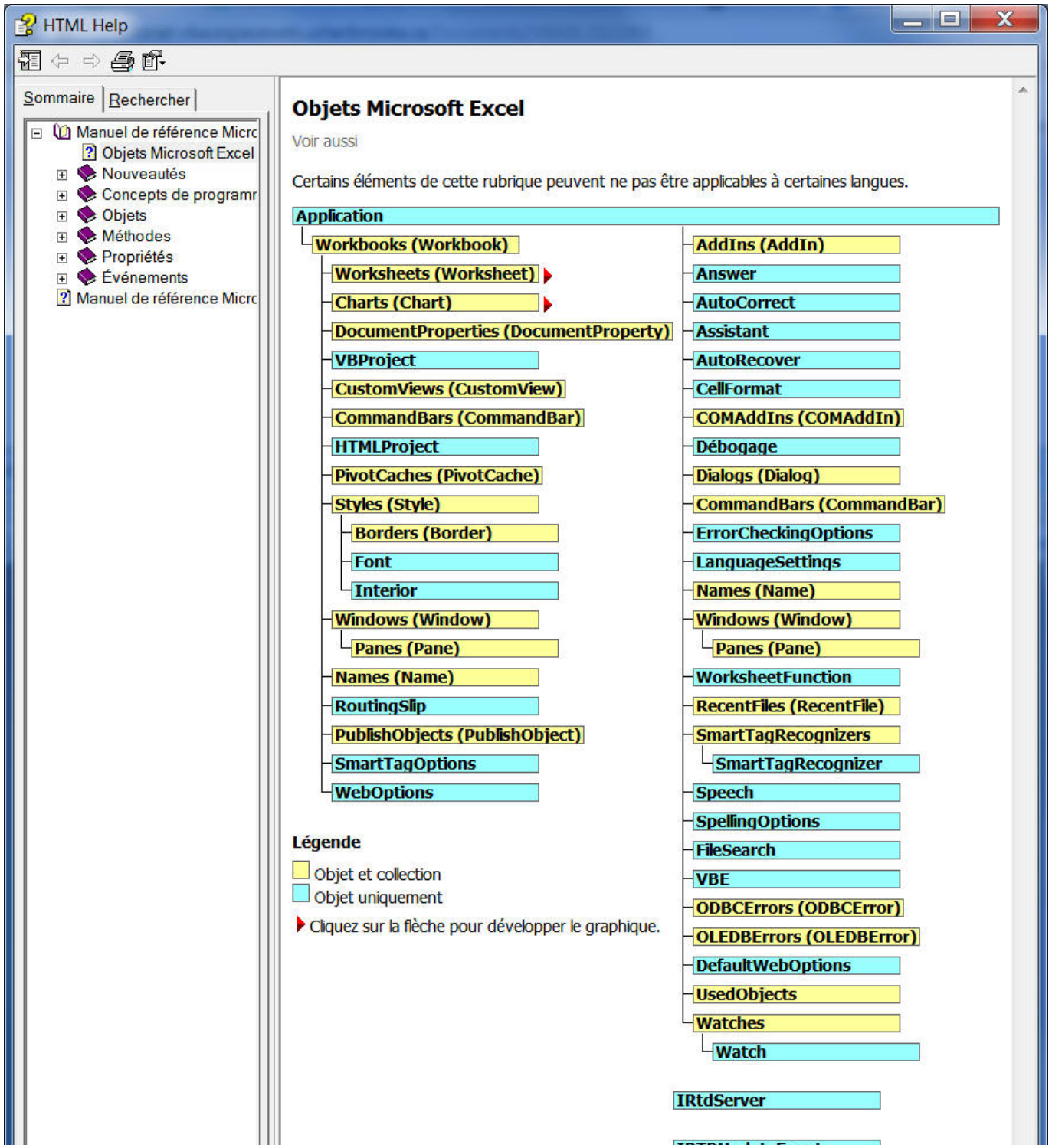
Quelques fichiers d'aide VBA sont disponibles en suivant les instructions d'installation suivantes:

Installer l'Aide VBA locale

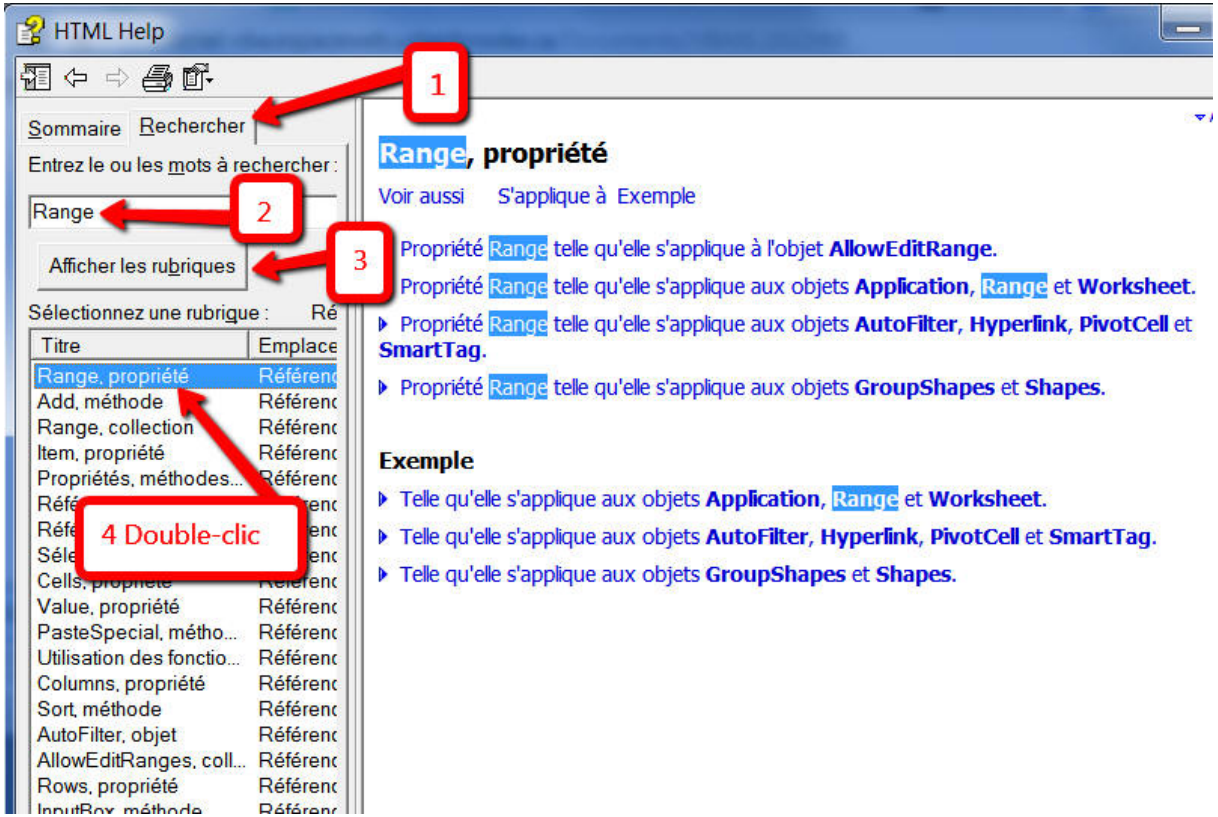
Vous y trouverez en particulier

- Un fichier d'aide pour l'environnement de développement VBE
- Un fichier d'aide pour la syntaxe VBA

- Un fichier d'aide pour le modèle objet Excel:



Vous pouvez aussi (et surtout!) utiliser l'onglet recherche pour obtenir de l'aide sur un objet Excel:



Raccourcis utiles en VBE

Ctrl + [ ] active Compléter mot.

F7 affiche la fenêtre code.

F2 affiche l'explorateur d'objets.

Ctrl + ↓ procédure suivante.

Ctrl + Page Down écran suivant.

Ctrl + C, Ctrl + V, ... Les autres raccourcis habituels fonctionnent aussi.

Tab

F9 ajoute/supprime un point d'arrêt.

Ctrl + Shift + F9 supprime tous les points d'arrêt.

F5 exécute la procédure active.

Ctrl + Pause Break interrompre l'exécution d'un module. Si votre clavier ne comporte pas de touche Pause/Break, contactez le fabricant de l'ordinateur pour trouver la combinaison de touches équivalente ou utilisez un autre clavier. Cette touche est indispensable pour déboguer les programmes.

F8 démarre le mode pas à pas.

Ctrl + F8 exécute jusqu'au curseur.

Ctrl + S enregistre le code VBA dans le classeur (mais n'enregistre pas le classeur!).

Alt + F11 bascule entre Excel et VBE.

Compléter le mot

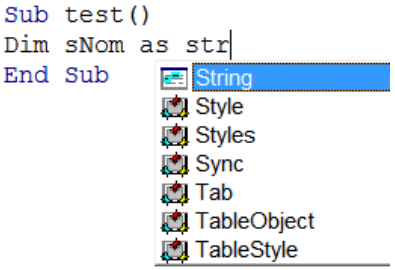
VBE offre une fonctionnalité qui complète un mot dont vous avez tapé les premières lettres.

Exemple: dans une procédure Sub tapez
Dim sNom as st

puis enfoncez les touches Ctrl + [ ]

VBE affiche la liste des mots commençant par st qui sont valides dans le contexte.

Si vous tapez une lettre additionnelle (r), la liste se raffine.



Et si la liste ne comporte qu'une seule possibilité, VBE complète le mot automatiquement.

Tapez Dim sPrénom as str

puis enfoncez les touches 

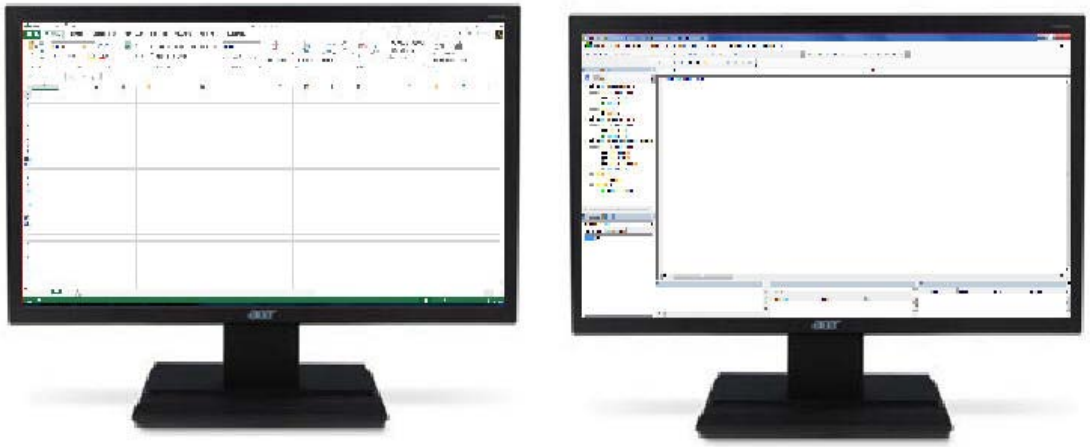
VBE complètera automatiquement

puisque String est le seul mot VBA commençant par Str qui est valide à cet endroit.

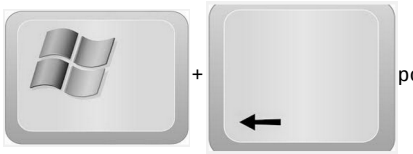
Dim sPrénom as String

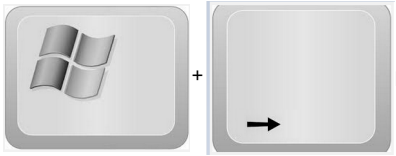
Organiser l'affichage

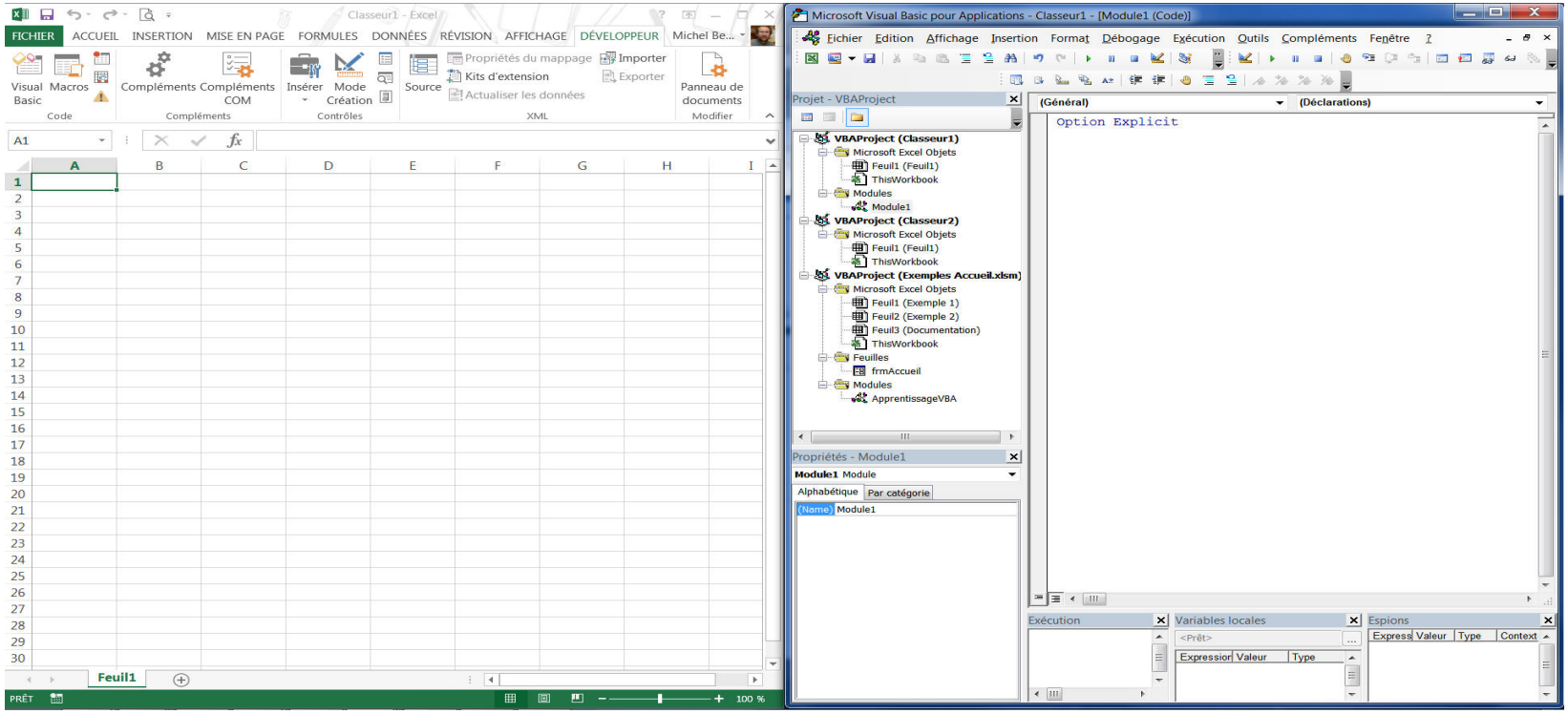
Lorsque vous travailles en VBA-Excel, vous travaillez en gfait avec 2 logiciels, chacun affichant sa fenêtre.
Si vous disposez de 2 écrans, vous devriez afficher la fenêtre Excel dans un écran, la fenêtre VBE dans l'autre:



Si vous ne disposez que d'un seul écran, utilisez les touches

 pour placer une fenêtre à gauche et

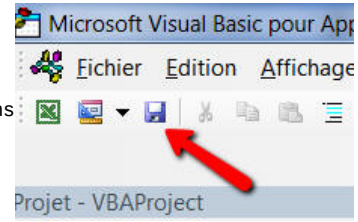
 pour placer une fenêtre à droite:



Dans tous les cas, vous pouvez utiliser les touches Alt + F11 pour basculer d'une application à l'autre.

Enregistrer un projet VBE

Pour enregistrer votre projet VBA, cliquez la disquette de la barre de boutons



Ou la combinaison de touches Ctrl + S

Ou encore fermer la fenêtre, tout simplement.

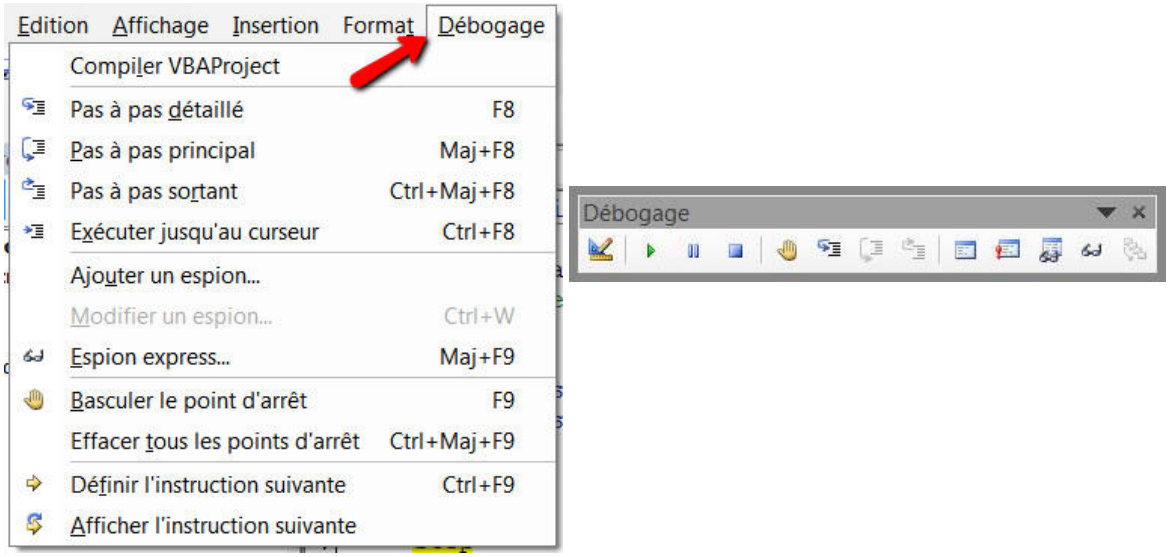
Mais **ATTENTION** votre projet VBA n'est pas sauvegardé sur disque, mais est sauvegardé dans le classeur Excel.

VOUS DEVEZ SAUVEGARDER LE CLASSEUR EXCEL POUR QUE VOTRE PROJET VBA SOIT SAUVEGARDÉ.

Déboguer

Un programmeur passe la grande majorité de son temps à tester et à déboguer du code. Il est donc essentiel de maîtriser les techniques de débogage ci-dessous.

VBE fournit un menu et une barre de boutons pour suivre l'exécution d'un programme VBA en mode pas à pas. De plus, la fenêtre Variables locale permet de suivre les valeurs des variables. Finalement, On peut suivre dans Excel les changements que le programme VBA apporte au clsseur.



Les principales techniques de débogage sont illustrées dans le document suivant:

Déboguer un programme VBA

Sommaire

L'environnement d'édition VBA (Visual Basic Editor) offre plusieurs fonctionnalités pour faciliter la rédaction de programmes VBA. Les principales sont:

- La fenêtre code
- La fenêtre variable locale
- Option Explicit
- Compléter le mot
- Les outils de débogage

Pour sauvegarder un projet VBA Excel, il faut sauvegarder **LE CLASSEUR Excel** qui le contient.

Dans cette section nous avons présenté les modes d'emploi suivants:

Modifier l'interface VBE

Installer l'Aide VBA locale

Déboguer un programme VBA

Fichier exemples.

Le fichier Excel Tutoriel Exemples VBE.xlsm contient l'exemple présenté dans la présente section.

Suite: Déclarations, types et références

VBA Excel

par Michel Berthiaume

http://www.usherbrooke.ca/adm/departements/simqg/professeurs/michel-berthiaume/

Déclarations, types et références

Sur cette page...

Les prochaines pages...

Un programme VBA manipule des valeurs. Ces valeurs peuvent prendre pluiseurs formes, chacune d'entre elle devant respecter des règles.

Valeurs littérales

Un littéral est une valeur explicitement tapée dans le programme.

Un littéral texte est entre guillemets:

- "Rouge"
- "Maman est chez le coiffeur"
  On peut inclure un apostrophe dans un littéral texte sans que la partie à droite soit considérée comme un commentaire:
  "L'apprentissage VBA"
  Pour inclure des guillemets dans un littéral texte, doubler les guillemets:
  "Quel ""beau"" programme"

Un littéral date ou heure est entre croisillons #:

- #2009-01-01#
- #14:00:00#
- #2009-01-01 14:00:00#

Un littéral numérique est un nombre:

- 12
- 2.5
- 16009829287654

Attention: en VBA, le séparateur décimal est toujours le point (.), jamais la virgule (,), peu importe la façon dont est configuré Excel. Pour le passage de valeurs décimales entre les deux programmes, la conversion se fait automatiquement.

2,5 n'est donc **JAMAIS** une valeur numérique en VBA.

Constantes

Une constante est un littéral auquel on donne un nom:

Une constante doit être déclarée et initialisée.
Par exemple, dans l'instruction
Const cDouzaine **=** 12

- **Const** est l'instruction qui indique à VBA que le texte qui suit est une déclaration de constante.
- cDouzaine est le nom de la constante.
- **=** est l'instruction d'assignation.
- 12 est une constante littérale qui sera copiée dans cDouzaine.

La valeur d'une constante **NE PEUT PAS** être modifiée lors de l'exécution du programme. On utilise donc les constantes pour donner un nom à des valeurs qui ne changent pas, comme le nombre d'objets dans une douzaine, la valeur de Pi, le nombre de secondes dans une journée.

VBA contient plusieurs constantes intrinsèques qui nomment des valeurs spécifiques à VBA.

Par exemple, **vbRed** vaut 255.

Les constantes intrinsèques sont automatiquement déclarées et initialisées par VBA. Pour en consulter la liste, utilisez l'explorateur d'objets:

- Dans l'Éditeur VBA, appuyez sur la touche 

(ou menu **Affichage**, choix **Explorateur d'objets**)
- Dans la 2e liste déroulante (1), tapez **constants**

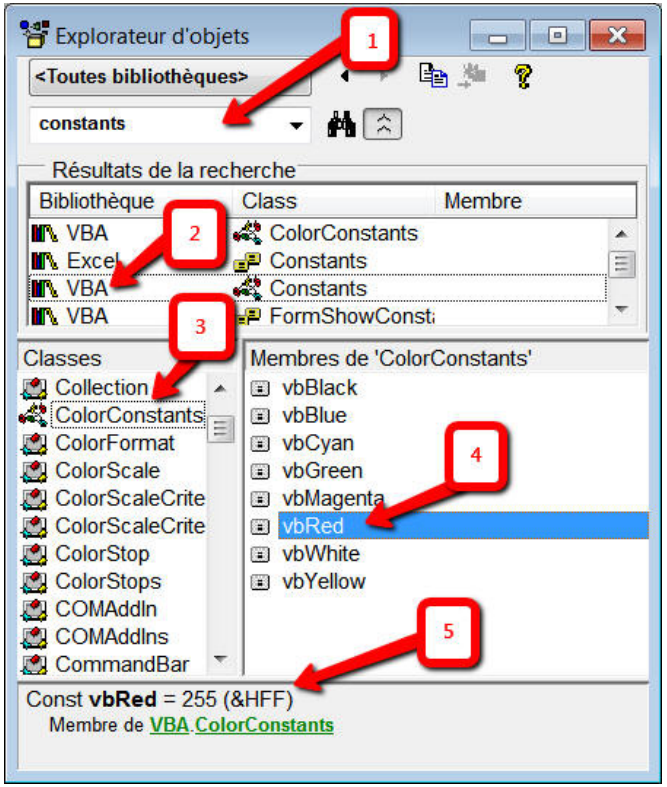
1) Critère de recherche dans les bibliothèques d'objets VBA.

2) Liste des bibliothèques d'objets répondant au critère.

3) Classes d'objets dans la bibliothèque choisie en 2).

4) Objets de la classe choisie en 3).

5) Valeur de l'objet choisi en 4).



Vous pouvez aussi consulter l'aide VBA (recherchez **Constantes**).

Pour déclarer et initialiser vos constantes, utilisez l'instruction **CONST**

[**Public | Private**] **Const** *nom* [**As** *type*] **=** *expression*

Où

- *nom* est un nom respectant les règles VBA
- *type* est un type VBA autre qu'objet
- *expression* est une expression VBA n'utilisant que des constantes ou des littéraux

Variables

Les littéraux et les constantes ont des valeurs fixes pendant toute l'exécution d'un programme VBA.

Un programme VBA ne peut pas changer la valeur d'une constante.

Les valeurs modifiables par le programme se trouvent dans des variables ou dans des propriétés d'objet.

Les variables sont des unités de stockage internes au programme VBA, alors que les propriétés d'objet sont des unités de stockage externes.

Par exemple, si votre programme doit compter le nombre de cellules jaunes d'une plage, il utilisera une variable.
Mais s'il veut changer la couleur d'une cellule, il utilisera la propriété color de l'objet de la classe Range contenant la cellule.

Lors de l'évaluation d'une expression (pendant exécution du programme), le nom de chaque variable est remplacé par son contenu.

Une variable est caractérisée par son nom et son type.

Choisir un nom

Un nom de variable VBA **doit**:

- Commencer par une lettre.
- Contenir un maximum de 255 lettres et/ou chiffres.

Un nom de variable **peut**

- Commencer par une lettre qui identifie son type (pratique **recommandée**).
- Décrire son contenu (pratique **recommandée**).
- Se terminer par un identificateur de type %, &, #, !, @ ou $ (pratique **déconseillée**).

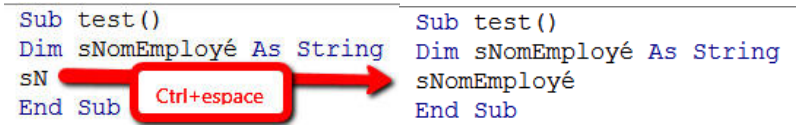
Un nom de variable **ne doit pas**

- Être identique à un mot réservé VBA.
- Contenir des espaces ni - ni des caractères spéciaux (sauf les identificateurs de type à la fin du nom). On peut cependant utiliser _ pour indiquer un espace.
- Commencer par un chiffre.
- Être ambigu (porter le même nom qu'une autre variable ou constante).

Le choix de noms en programmation fait l'objet de plusieurs pratiques et conventions (voir
http://fr.wikipedia.org/wiki/Convention_de_nommage).
Dans le présent document, on a adopté la pratique de

- toujours choisir un nom français décrivant le contenu de la variable (ou constante)
- toujours précéder ce nom d'au moins une lettre décrivat le type de la variable:
  - sNomEmployé contient un nom dans une variable de type string
  - lCompteur contient un compteur de type Long
  - cMontant contient un montant de type Currency

Un des avantages de cette pratique se manifeste lorsqu'on utiliser la fonctionnalité Compléter le mot de VBE. En effet:



Comme il n'y a qu'un seul objet dont le nom commence par sN qui est valide à cet endroit, taper sN suivi de

complète le nom de la variable sans effort.

Il n'y a donc pas de problème à utiliser des noms très descriptifs (et longs), en autant que les premières lettres les distinguent de noms d'autres objets.

Dans ce contexte, il faut particulièrement éviter les nom de variable d'une seule lettre, qui n'ont aucune valeur descriptive, sauf les variables i, j et k utilisés dans la manipulation de tables, matrices et collections.

Déclarations

Une variable VBA doit être déclarée pour que VBA lui réserve un espace en mémoire centrale et lui assigne un nom. La quantité et l'organisation de l'espace réservé dépend du type de variable.

Si **Option Explicit** n'est pas présent au début du module VBA où se trouve une variable, et que la variable n'est pas déclarée, VBA la déclarera automatiquement de type Variant, qui peut contenir n'importe quoi.

Apparemment pratique, **cette fonctionnalité est très dangereuse,** car elle désactive la détection de fautes de frappe.

En effet VBA ne détectera pas d'erreur dans le code suivant:

```
Sub Exemple()
Dim cCompteur As Currency

   cComteur = cCompteur + 10
End Sub
```

Mais la détectera si vous faites plutôt:

```
Option Explicit

Sub Exemple()
Dim cCompteur As Currency

   cComteur = cCompteur + 10
End Sub
```

Pour automatiser l'insertion de Option Explicit au début de chaque module:

- Dans le menu **Outils**, choisissez **Options**
- Dans l'onglet **Éditeur**, assurez-vous que la case à cochée **Déclaration des variables obligatoire** est cochée.

Voir les modifications recommandées à l'interface VBE:

Modifier l'interface VBE

Une variable est explicitement déclarée dans les instructions **Dim**, **Private**, **Public**, **Function** et **Sub**.

Dim *nom*[**(**[*indices*]**)**] [**As** [**New**] *type*] [**,** *nom*[**(**[*indices*]**)**] [**As** [**New**] *type*]] **. . .**
Private *nom*[**(**[*indices*]**)**] [**As** [**New**] *type*] [**,** *nom*[**(**[*indices*]**)**] [**As** [**New**] *type*]] **. . .**
Public *nom*[**(**[*indices*]**)**] [**As** [**New**] *type*] [**,** *nom*[**(**[*indices*]**)**] [**As** [**New**] *type*]] **. . .**

où

- *nom* est un nom respectant les règles VBA.
- *indices* déclare les dimensions d'un tableau.
- **NEW** valide seulement si *type* est un objet, déclenche la création de l'objet.
- *Type* est un type VBA ou une classe d'objets. Par défaut le type de données est Variant.

Dim peut être utilisé:

- Au début d'un module (en dehors d'une procédure). Déconseillé.
- Au début d'une procédure. Recommandé.
- À l'intérieur d'une procédure. Fortement déconseillé, parce que tous les **Dim** d'une procédure sont de toute façon exécutés avant les autres instructions du module, et leur présence ailleurs qu'au début donne l'impression du contraire.

Les valeurs numériques sont initialisées à 0, les valeurs texte à "", les dates à #00:00:00#, les variant à **Empty** et les objets à Nothing.

Noter que l'instruction Dim utilisée à l'intérieur d'une procédure Sub ou Function initialise toujours la valeur de la variable à sa valeur par défaut, peu importe l'endroit où se trouve Dim danbs la procédure.

Autrement dit, la valeur des variables déclarées dans une procédure sont perdues dès que la procédure est terminée.

Les instructions **Private** et **Public** ont la même syntaxe que **Dim**, mais ils ont une portée différente.

Les instructions Function et Sub permettent aussi de déclarer des variables, qui seront les paramètres de la procédure.

Types

Type de données	Description	Plage	Valeur initiale	Suffixe (utilisation déconseillée)
Byte	Un octet	0 à 255	0	
Boolean	Vrai ou faux, Oui ou Non	**True** ou **False**	False	
Integer	Un nombre entier	-32 768 à 32 767	0	%
Long	Un nombre entier	-2 147 483 648 à 2 147 483 647	0	&
LongLong	Un nombre entier. Ce type n'est disponible que depuis Office 2010 et uniquement dans les versions 64 bits.	-9,223,372,036,854,775,808 à 9,223,372,036,854,775,807	0	^
Single	Un nombre pouvant contenir une partie décimale. La valeur peut être une approximation de la valeur assignée.	-3,402823E38 à -1,401298E-45 pour les valeurs négatives ; 1,401298E-45 à 3,402823E38 pour les valeurs positives	0	!
Double	Un nombre pouvant contenir une partie décimale. La valeur peut être une approximation de la valeur assignée.	-1,79769313486231E308 à -4,94065645841247E-324 pour les valeurs négatives ; 4,94065645841247E-324 à 1,79769313486232E308 pour les valeurs positives	0	#
Currency	Un nombre pouvant contenir jusqu'à 4 chiffre en partie décimale.	-922 337 203 685 477,5808 à 922 337 203 685 477,5807	0	@
Date	Une date et une heure	1er janvier 100 au 31 décembre 9999	#00:00:00#	
Object		Toute référence à des données de type **Object**	**Nothing**	
String	Du texte	0 à environ 2 milliards de caractères	"" ou **vbNullString**	$
Variant	Le type indéfini. VBA détermine le type du contenu d'une variable Variant lorsque le programme lui assigne une valeur.		**Empty**	
Type défini par l'utilisateur	Type défini par l'utilisateur. Voir **DefType dans l'aide VBA**			

VBA offre une grande gamme de types de données, pour être compatible avec des programmes écrits dans des versions antérieures du langage.
En pratique, évitez d'utiliser les types:

- **Byte** : réservé aux programmeurs chevronnés qui manipulent le contenu des 8 bits d'un octet.
- **Integer** : utilisez **currency**, meilleure capacité et plus souple.
- **Long**: utilisez **currency**, meilleure capacité et plus souple.
- **Single**: utilisez **currency**, qui ne fait pas d'arrondi.
- **Double**: utilisez **currency**, qui ne fait pas d'arrondi.
- **Variant**: désactive la validation de type de VBA.

Donc, pour stocker

- un nombre, utilisez **Currency**.
- du texte, utilisez **String**.
- une date, utilisez **Date**.
- une heure, utilisez **Date**.

Vous devrez utiliser les autres types lorsqu'ils sont utilisés dans des programmes existants ou dans les propriétés d'objets.

Dans le cas particulier des paramètres de Function ou Sub paramétré, il est préférable de les déclarer **Variant** puis de valider le contenu dans la procédure.
On évite ainsi des messages d'erreur déroutants lorsque les paramètres sont mal initialisés. On verra dans la section Tests et branchements comment gérer ces paramètres.

Valeurs spéciales

En plus des valeurs "normales", comme du texte dans une variable String, un nombre dans une variable Currency, les variables peuvent avoir des valeurs "spéciales" dans des situations "spéciales".

Empty: une variable de type Variant est initialisée à **Empty** lors de sa création.

- **IsNull**(*nom*) est **Faux** lorsque *nom* est **Empty**.
- **IsEmpty**(*nom*) est **Vrai** lorsque *nom* est **Empty**
- *nom* **=** *0* est **Vrai** lorsque *nom* est **Empty**
- *nom* **=** *""* est **Vrai** lorsque *nom* est **Empty**

"" (vbNullString ou vbNullChar): une valeur de type String vide.

- **IsNull**(*nom*) est **Faux** lorsque *nom* **=** ""
- **IsEmpty**(*nom*) est **Faux** lorsque *nom* **=** ""
- *nom* **=** *0* est **Faux** lorsque *nom* **=** ""

Null: une variable **Null** n'existe pas. Aucun espace mémoire ne lui a été assigné. Seul son nom existe. S'applique le plus souvent à des propriétés d'objets.

- **IsNull**(*nom*) est **Vrai** lorsque *nom* est **Null**.
- **IsEmpty**(*nom*) est **Faux** lorsque *nom* est **Null**.
- *nom* **=** *0* est **Faux** lorsque *nom* est **Null**.
- *nom* **=** *""* est **Faux** lorsque *nom* est **Null**.

Nothing: lorsqu'une variable objet est déclarée mais pas assignée (avec **SET**). Ne s'applique qu'aux variables objet. Pas aux propriétés d'objet, pas aux variables VBA.

- **IsNothing**(*nom*) est **Vrai** lorsque *nom* est **Nothing**.

Conversion

VBA convertit automatiquement les valeurs de types différents au besoin. Il n'est donc pas nécessaire de s'assurer que toutes les variables soient du même type avant de les utiliser dans une expression. Il arrive cependant que VBA convertisse tout en entier, ce qui peut mener à la perte de décimales. Il est donc préférable d'utiliser le type **currency** pour toutes les variables qui s'y prêtent, soit en pratique toutes les variables numériques de vos programmes.

Le code VBA à droite fonctionne, car VBA convertit S10 en entier avant de l'ajouter à 1 et mettre le résultat dans cCompteur.

```
Sub Exemple()
Dim cCompteur As Currency
Dim s10 As String
    s10 = "10"
    cCompteur = s10 + 1
End Sub
```

Mais le code à droite ne fonctionnera pas et se terminera par un message d'erreur "Incompatibilité de type"

```
Sub Exemple()
Dim cCompteur As Currency
Dim s10 As String
    s10 = "dix"
    cCompteur = s10 + 1
End Sub
```

Il n'est pas prudent de se fier à la conversion automatique de types de VBA. Si elle fonctionne **PRESQUE** tout le temps, il arrive que des données imprévues donnent des résultats inattendus.

VBA fournit donc des Fonctions de conversion: **Val**(), **Str**(), **Format**(), **FormatCurrency**(), **FormatDateTime**(), **CBool(), CByte(), CCur(), CDate(), CDbl(), CDec(), CInt(), CLng(), CSng(), CStr(), CVar()**

Val(*expression***)**

expression peut être n'importe quelle expression de type string.

Val() tente d'extraire un nombre de l'expression. **Val()** est l'opposé de **Str()**.

Où

- Si l'expression ne contient que des chiffres, **Val(expression)** retourne un nombre composé de ces chiffres.
- Si expression ne contient aucun chiffre, **Val (expression)** retourne 0.
- Si l'expression contient un mélange de chiffres et de lettres, Val(expression) fait de son mieux en arrêtant la récupération des chiffres au premier caractère alphabétique.

Point décimal: **Val**() ne reconnaît que le point comme séparateur décimal. Si vous voulez permettre l'utilisation d'un autre symbole (comme la virgule) vous devrez utiliser l'instruction **Replace()** au préalable ou utiliser **Ccur()** pour faire la conversion.

Str(*expression*)

expression peut être n'importe quelle expression **sauf de type string**

Str(*expression***)** convertit en string le contenu de *expression.* **Str()** est l'opposé de **Val()**.

Le format est choisi par VBA et peut ne pas convenir (séparateur de décimales, format de date ou d'heure...).
Vous pouvez préférer l'utilisation de la fonction **Cstr()** ou, mieux encore, d'une des fonctions **Format(), FormatCurrency(), FormatNumber() ou FormatDateTime()**

Format(*expression*[, *format*[, *premierjour*[, *premièresemaine*]]])

où

- *expression* est une expression VBA.
- *format* est une expression de type string décrivant le format voulu
  voir les règles ci dessous
- *premierjour* expression entre 1 et 7 indiquant le premier jour de la semaine (1 = dimanche).
- *premièresemaine*: 1 (semaine du 1er janvier), 2 (première semaine d'au moins 4 jours dans la nouvelle année) ou 3 (première semaine complète).

format peut être

- un format numérique prédéfini:
  - Nombre général.
  - Monétaire.
  - Fixe.
  - Standard.
  - Pourcentage.
  - Scientifique.
  - Oui/Non: Non si expression vaut 0 sinon Oui.
  - Vrai/Faux: Faux si expression vaut 0 sinon Vrai.
  - Actif/Inactif: Inactif si expression vaut 0 Actif.
- un format date ou heure prédéfini (suivant les paramètres Windows)
  - Date, général
  - Date, complet
  - Date, réduit
  - Date, abrégé
  - Heure, complet
  - Heure, réduit
  - Heure, abrégé
- un masque de formatage.

- Consultez l'aide VBA (recherchez **format**) pour connaître les nombreuses possibilités et surtout de nombreux exemples.

FormatCurrency(*Expression*[,*NombreDécimales*[,*ZéroNonSignificatif*[,*Parenthèses*[,*Groupes*]]]])

FormatNumber(*Expression*[,*NombreDécimales*[,*ZéroNonSignificatif*[,*Parenthèses*[,*Groupes*]]]])

Où:

- *Expression* Expression à formater.
- *NombreDécimales* Positions à droite de la virgule sont affichées. Par défaut (-1) les paramètres Windows sont employés.
- *ZéroNonSignificatif* -1 les affiche, 0 ne les affiche pas, -2 utilise les paramètres Windows.
- *Parenthèses* -1 les affiche, 0 ne les affiche pas, -2 utilise les paramètres Windows.
- *Groupes* -1 groupe les milliers, 0 ne les groupe pas, -2 utilise les paramètres Windows.

Ces deux fonctions convertissent *Expression* en format **String** en utilisant les paramètres de présentation des paramètres régionaux de Windows.

FormatDateTime(*Expression*[,*Format*])

Où

- *Expression* Expression de type date
- *Format* une des valeurs suivantes:
  0 affiche la partie date sous forme de date abrégée et la partie heure sous forme d'heure complète
  1 Affiche la date en format de date complet spécifié dans les paramètres Windows
  2 Affiche la date en format de date abrégé spécifié dans les paramètres Windows
  3 Affiche l'heure en format d'heure spécifié dans les paramètres Windows
  4 Affiche l'heure au format 24 heures (hh:mm).

Cette fonction convertit *Expression* en format **String** en utilisant les paramètres de présentation des paramètres régionaux de Windows.

CBool(*expression***)**
CByte(*expression***)**
CCur(*expression***)**
CDate(*expression***)**
CDbl(*expression***)**
CDec(*expression***)**
CInt(*expression***)**
CLng(*expression***)**
CSng(*expression***)**
CStr(*expression***)**
CVar(*expression***)**

Où

- *expression* peut être de n'importe quel type, mais dont les valeurs acceptables varient selon le type de conversion demandée. En effet, la conversion doit être possible.

Ces fonctions convertissent explicitement *expression* dans le type indiqué par le nom de la fonction. Si la conversion est impossible, comme dans
CCur("rouge")
l'exécution du programme est interrompue avec le message "Incompatibilité de type".
Pour la fonction **CCur**, *expression* doit valoir entre -922 337 203 685 477,5808 et 922 337 203 685 477,5807.

Voir la section Gestion des erreurs pour une technique permettant de contourner le problème potentiel.

Portée de variables

En principe, une variable déclarée **dans** une procédure **Sub** ou une procédure **Function** ne peut être utilisée que dans cette procédure:

Ici, cMontant est une variable locale de la procédure Exemple

```
Sub Exemple2()
Dim cMontant As Currency
    InputBox cMontant
End Sub
```

Une variable déclarée **au début d'un module** (avant la première procédure) peut être utilisée dans toutes les procédures du module:

Ici, cMontant3 est une variable locale du module et publique des procédures du module.

```
Dim cMontant3 as currency
Sub Exemple3()
    InputBox cMontant3
End Sub
Sub AutreSub()
    cMontant3 = cMontant3 + 10
End Sub
```

Pour qu'une variable puisse être utilisée par toutes les procédures de tous les modules, il faut la déclarer en utilisant l'instruction **Public** au lieu de l'instruction **Dim**.

Il est aussi possible d'éviter qu'une variable locale perde sa valeur à la fin de l'exécution de la procédure dont elle fait partie en utilisant l'instruction **Static** au lieu de **Dim**. Il est préférable de stocker ces valeurs dans des cellules Excel.

Recommandation: n'utilisez que des variables locales de procédure. Utilisez les paramètres de **Sub** ou de **Function** ou, mieux encore, des cellules Excel pour partager des valeurs entre les procédures et les modules.

Exemples:

À la fin de l'exécution de la procédure **Sub** Principale ci-contre, la variable cTaux contiendra 0.15.

```
Option Explicit
Dim cTaux As Currency
Sub Principale()
   cTaux = 0.05
   Call NouveauTaux
   MsgBox cTaux
End Sub
Sub NouveauTaux()
   cTaux = 0.15
End Sub
```

À la fin de l'exécution de la procédure **Sub** Principale2 ci-contre, la variable cTaux contiendra 0.05.

```
Option Explicit
Sub Principale2()
Dim cTaux As Currency
   cTaux = 0.05
   Call NouveauTaux2
   MsgBox cTaux
End Sub
Sub NouveauTaux2()
Dim cTaux As Currency
   cTaux = 0.15
End Sub
```

La procédure Sub ci-contre affiche successivement:

2

3.00

3

Alors que si t1 et t2 sont déclarés **Currency**, elle affichera

3

3.00

3

```
Sub sDivision()
Dim Taux1 As Double
Dim Taux2 As Double
   Taux1 = 0.15
   Taux2 = 0.05

    MsgBox Int(Taux1 / Taux2)
   MsgBox Format(Taux1 / Taux2, "###.00")
   MsgBox Format(Taux1 / Taux2, "###")
End Sub
```

Sommaire

- En VBA il n'est pas nécessaire de déclarer les variables avant de les utiliser. Mais comme c'est une très mauvaise pratique, on s'assure de commencer chaque module par l'instruction Option Explicit qui désactive cette fonctionnalité.
- Parmi les nombreux types disponibles pour les variables, on se limite à utiliser Currency, String, Date et à l'occasion Long.
- Le choix d'un nom de variable a une grande influence sur la facilité à comprendre (donc à modifier) un programme.
- En VBA Excel, on utiliser toujours des variables locales.
- On se méfie de la fonction conversion automatique des nombres en VBA. Une autre bonne raison d'éviter les types autre que currency (pour les nombres).

Dans cette section nous avons présenté le mode d'emploi suivant:

 Modifier l'interface VBE

Fichier exemple.

Le fichier Excel Tutoriel Exemples Déclarations.xlsm contient les exemples présentés dans la présente section.

Suite: Expressions et assignations

VBA Excel

par Michel Berthiaume

http://www.usherbrooke.ca/adm/departements/simqg/professeurs/michel-berthiaume/

Expressions et assignations

Sur cette page...

Les prochaines pages...

Instruction d'assignation

Il y a plusieurs façons d'assigner une valeur à une variable. Les plus utilisées en VBA sont les instructions d'assignation **Let** et **Set**:

[Let] *nom* = *expression*

où:

- **Let** Le nom de l'instruction. **Let** est pratiquement toujours omis.
- *nom* Nom de la variable VBA ou de la propriété d'objet. *nom* ne doit pas désigner un objet.
- *expression* Une expression VBA dont le résultat sera attribué à *nom*. *Expression* ne doit pas être un objet.

Cette instruction d'assignation évalue *expression* et assigne la valeur résultante à la variable *nom* en faisant les conversions nécessaires selon les types des variables utilisées dans *expression*. Donc la valeur de l'*expression* est **COPIÉE** dans *nom*.

Attention au symbole = qui n'a pas le même sens entre *nom* et *expression* et dans *expression*.

Set *nom* = {[**New**] *expression* | **Nothing**}

- *nom* Nom de la variable ou propriété d'objet qui contiendra l'objet.
- **New** Crée une nouvelle instance de la classe identifiée par *expression*.
- *expression* Représentant le nom d'un objet, d'une autre variable déclarée du même type objet

ou d'une fonction ou méthode renvoyant un objet du même type ou d'une classe d'objets.
- **Nothing** Efface la variable *nom* et libère les ressources qui étaient associées.

Si **New** est absent, cette instruction d'assignation évalue *expression* et donne le nom *nom* à l'objet résultant. Donc *nom* et *expression* désignent **LE MÊME** objet.

Si **New** est présent, cette instruction crée un nouvel objet de la classe d'*expression* nommé *nom*. Donc *nom* et *expression* désignent deux objets **DIFFÉRENTS**.

Il y a 2 grandes différences entre Let et Set:

- Let (assignation VBA) copie la valeur de l'expression.
- Dans Let, la valeur de l'expression doit être d'un type VBA.
- Set (assignation objet) donne un nom à l'expression. Ce n'est pas une copie.
- Dans Set, la valeur de l'expression doit être un objet.

D'autres instructions peuvent aussi assigner une valeur à une variable:

For, **InputBox**, **Input**, **Read**, **Get**, ...

Expressions

Comme dans tout langage de programmation, les expressions VBA sont composées de variables, de propriétés d'objet, de constantes et/ou de fonctions, liées par des opérateurs et des parenthèses.

Tout ce que VBA peut évaluer est une expression et peut faire partie d'une expression.

Les règles sont presque identiques à celles des formules Excel.

Le résultat de l'évaluation d'une expression est la valeur de l'expression, dont le type dépend des éléments composant l'expression.
La valeur d'une expression peut être **Null**.

ATTENTION: lorsqu'on mélange les types à l'intérieur d'une expression, VBA convertit automatiquement les valeurs dans les types qui SEMBLENT les plus appropriés. Si la plupart du temps, le résultat est correct, il ne l'est pas toujours.

VBA ne détecte aucune erreur
dans le programme à droite et
il s'exécute correctement.
	`Sub exemple1()`
	`Dim sNombre As String`
	`  sNombre = 22`
	`  sNombre = sNombre + 2`
	`End Sub`

VBA ne détecte aucune erreur dans le programme
à droite mais une erreur
"Incompatibilité de type" se produit à l'exécution
	`Sub exemple2()`
	`Dim sNombre As String`
	`  sNombre = "Ving deux"`
	`  sNombre = sNombre + 2`
	`End Sub`

Il est beaucoup plus prudent de vous assurer que les types des éléments de l'expression soient explicites.

```
Sub exemple()
Dim sNombre As String

    sNombre = 22
    If IsNumeric(sNombre) Then
        sNombre = CCur(sNombre) + 2
    End If

End Sub
```

```
Sub exemple3()
Dim cNombre As Currency

    cNombre = 22
    cNombre = cNombre + 2

End Sub
```

ou utilisez les fonctions de conversion.

Opérateurs

Dans les expressions VBA, vous pouvez utiliser les opérateurs suivants:

Opérateurs arithmétiques
Ces opérateurs combinent deux expressions numériques et retournent une expression numérique.

+	Addition
–	Soustraction
*	Multiplication
/	Division
\	Division entière: retourne un entier
MOD	Reste de division
^	Exposant

Opérateurs de comparaison
Ces opérateurs comparent deux expressions numériques et retournent Vrai ou Faux (type Boolean).

=	Égal à
>	Supérieur à
<	Inférieur à
>=	Supérieur ou égal à
<=	Inférieur ou égal à
<>	Différent de

Opérateurs de concaténation de texte
Ces opérateurs comparent deux expressions **String** et retournent une expression String.

& combiner (concaténer) deux **Strings**
+ combiner (concaténer) deux **Strings** ou en additionne les valeurs. L'utilisation de + pour concaténer deux expressions crée une confusion inacceptable et doit être évitée à tout prix.

Opérateurs logiques
Ces opérateurs retournent une expression Boolean.
Pour tester si une expression vaut Null, utiliser la fonction **IsNull**().
Si une des expressions n'est pas de type Boolean, VBA convertit l'expression de la façon suivante:
0 donne Faux, et toute autre valeur donne Vrai (**Not** Faux).

And	(Vrai **AND** Vrai) donne Vrai (Vrai **AND** Faux) donne Faux (Vrai **AND Null**) donne **Null** (Faux **AND** Vrai) donne Faux (Faux **AND** Faux) donne Faux (Faux **AND Null**) donne Faux (Null **AND** Vrai) donne **Null** (Null **AND** Faux) donne Faux (Null **AND Null**) donne **Null**
Or	(Vrai **OR** Vrai) donne Vrai (Vrai **OR** Faux) donne Vrai (Vrai **OR Null**) donne Vrai (Faux **OR** Vrai) donne Vrai (Faux **OR** Faux) donne Faux (Faux **OR Null**) donne **Null** (Null **OR** Vrai) donne Vrai (Null **OR** Faux) donne **Null** (Null **OR Null**) donne **Null**
Xor	(Vrai **XOR** Vrai) donne Faux (Vrai **XOR** Faux) donne Vrai (Vrai **XOR Null**) donne **Null** (Faux **XOR** Vrai) donne Vrai (Faux **XOR** Faux) donne Faux (Faux **XOR Null**) donne **Null** (Null **XOR** Vrai) donne **Null** (Null **XOR** Faux) donne **Null** (Null **XOR Null**) donne **Null**
Not	(**NOT** Vrai) donne Faux (**NOT** Faux) donne Vrai (**NOT Null**) donne **Null** **IsNull(Null)** donne Vrai **NOT IsNull(Null)** donne Faux

Eqv	Voir l'aide VBA
Imp	Voir l'aide VBA

ATTENTION: Contrairement à ce qu'on peut penser, (**NOT Null**) retourne **Null**.
Pour tester si une expression vaut **Null**, on utilise le plus souvent
IF NOT IsNull(*expression*) **THEN** ...

L'opérateur TypeOf retourne aussi une valeur Boolean:

TypeOf *variable* **Is** *expression*

Où

- *variable* est le nom d'une variable objet
- *expression* est un nom de classe

TypeOf vaut Vrai si *variable* est de type *expression* et vaut Faux dans les autres cas. Voir **TypeName()**

Opérateur Like

variable **Like** *expression*

Où

- *variable* est le nom d'une variable de type String
- *expression* est une expression de type String contenant un masque de comparaison.

La comparaison est évaluéa à Vrai si la valeur String à gauche de Like correspond au masque à droite.

Like permet de vérifier si la valeur d'une variable correspond à un masque.

Les principaux caractères passe-partout reconnus dans le masque sont:

? qui remplace un caractère

* qui remplace 0, un ou pulsieurs caractères

qui remplace un chiffre

Exemples:

Si sTexte contient		Vaut
Marie	sTexte Like "M*"	Vrai
Marie	sTexte Like "M?"	Faux
J1K 2R1	sTexte Like "?#? #?#"	Vrai
J1K 2R1	sTexte Like "?#?#?#"	Faux
123 456	sTexte Like "?#? #?#"	Vrai

Il existe plusieurs autres caracéristiques de l'opérateur Like qu'il serait trop long de décrire ici. Consulter au besoin l'aide Microsoft VBA sur les sujets **Like** et **Option compare**.

Parenthèses

Dès qu'une opération comporte plus d'un opérateur, VBA applique l'ordre de priorioté suivant :

^

- (négation)

*, /

\

Mod

+,- (addition, soustraction)
&
=
< >
 <
>,
< +
> =
Not
And
Or
Xor
Eqv
Imp

Pour éviter toute confusion, il est préférable d'utiliser des parenthèse lorsqu'il y a plus d'un opérateur dans une expression, même si l'ordre de priorité est respecté. Ici, la prudence ne coûte rien.

Fonctions

VBA permet l'utilisation de fonctions dans les expressions VBA.

On peut utiliser des fonctions VBA (souvent semblables à des fonctions Excel) ou même des fonctions Excel.

Voici quelques fonctions VBA fréquemment utilisées:

- Fonctions retournant une valeur de type string:
  - Left()
  - Right()
  - Mid()
  - Trim()
  - InputBox
  - Format()
  - MonthName()
  - Replace()
  - WeekdayName()
- Fonctions retournant une valeur numérique:
  - InStr()
  - Len()
  - Int()
  - DateDiff()
  - Day()
  - Month()
  - Year()
  - LBound()
  - UBound()
  - MsgBox()
  - Rnd()
  - Round()
  - Val()
- Fonctions retournant une valeur de type Date:
  - Date()
  - Now()
  - DateAdd()
  - DateSerial()
  - DateValue()
- Autres fonctions:
  - EOF()
  - IIf()

- TypeName()

Voici comment utiliser la fonction Excel Max dans VBA:

```
WorksheetFunction.Max(Un, Deux)
```

Notez que l'exécution d'une fonction Excel à partir de VBA est peu performante et qu'il est nettement préférable d'utiliser la fonction VBA équivalente lorsqu'elle existe.

S'il faut utiliser une fonction Excel, il est préférable de l'utiliser dans une cellule Excel et d'aller lire le résultat dans la cellule.
Soit inscrire
=Max(A1;B1)
dans une cellule (disons C1). On peut alors utiliser Range("C1") pour avoir le maximum de A1 et B1.

Exemples

Assignation	Si	cRésultat vaut
cRésultat = 3		3
cRésultat = 3 + 4		7
cRésultat = A + 4	A contient 3	7
cRésultat = Résultat + 1	cRésultat contient 7	8
cRésultat = A = 3	A contient 3	-1 (équivalent à vrai)
cRésultat = A = 4	A contient 4	0 (équivalent à vrai)
cRésultat = 2 + 3 * 5		17
cRésultat = (2 + 3) * 5		25

Le fichier Exemples Expressions.xlsm contient ces exemples et plusieurs autres, dans le module VBA Expressions.

Sommaire

Les expressions VBA suivent sensiblement les mêmes règles que Excel, en utilisant des variables VBA plutôt que des cellules.

Il existe un bon nombre d'opérateurs et de fonctions qu'on peut utiliser dans une expression Excel.

Dès qu'on a plus d'un opérateur dans une expression, mieux vaut utiliser des parenthèses.

L'instruction d'assignation régulière évalue l'expression à droite du signe = puis en copie la valeur dans la variable à gauche. Exemple: cRésultat = A copie la valeur de A dans cRésultat.

L'instruction d'assignation SET identifie l'objet résultant de l'évaluation de l'expression à droite du signe = puis lui assigne le nom à gauche. Exemple: Set FeuilleDestination = Worksheets("Documentation") assigne la variable VBA FeuilleDestination à la feuille Documentation du classeur. Il n'y a pas de copie.

Fichier exemples

Le fichier Excel Tutoriel Exemples Expressions.xlsm contient les exemples présentés dans la présente section.

Suite: Tests

VBA Excel

par Michel Berthiaume

http://www.usherbrooke.ca/adm/departements/simqg/professeurs/michel-berthiaume/

Tests et branchements

Sur cette page...

Les prochaines pages...

Les instructions de test

Les instructions d'un programme sont normalement exécutées les unes à la suite des autres, séquentiellement.
Les instructions de test, de boucle et de branchement permettent de modifier cet ordre naturel.

If *expression* **Then**
  [*instructions*]
[**Else**
  [*instructions*]]
End If

Où

- *expression* Une expression VBA qui est évaluée à Vrai ou Faux. Si *expression* vaut **Null**, elle est considéré comme Faux.
- *instructions* Une ou plusieurs instructions VBA, sur autant de lignes que nécessaire.

La première série d'*instructions* est exécutée si *expression* vaut Vrai, et la seconde si *expression* vaut Faux

À la version de base ci-dessus qui ne permet que 2 alternatives, s'ajoutent les variations suivantes:

If *expression* **Then** *instructions*

Où

- *expression* Une expression VBA qui est évaluée à Vrai, Faux ou **Null**.
- *instructions* Une ou plusieurs instructions VBA séparées par des : qui seront exécutées si *expression* vaut Vrai

Il est fortement recommandé de ne JAMAIS utiliser cette forme de **IF** qui n'ajoute rien et rend le programme plus difficile à comprendre et plus difficile à modifier. Utilisez plutôt:

```
If expression Then
   [instructions]
End If
```

```
If expression Then
   [instructions]
[ElseIF
    [instructions]]
[ElseIF
    [instructions]]
    ...
[Else
   [instructions]]
End If
```

Où

- *expression* Une expression VBA qui est évaluée à Vrai ou Faux. Si *expression* vaut **Null**, elle est considéré comme Faux.
- *instructions* Une ou plusieurs instructions VBA

Chaque série d'*instructions* est exécutée si l'*expression* précédente vaut Vrai, sauf celle qui suit le **Else**, qui est exécutée quand aucune des précédentes n'est Vraie.

Cette variante du If est recommandée si votre programme doit tester plus d'une valeur d'une expression.

```
Select Case expression
   [Case liste/expression
      [instructions]] ...
   [Case Else
      [instructions]]
End Select
```

Où

- *expression* expression VBA
- *liste/expression* Liste de valeur(s) possibles(s) pour *expression ou expresseion VBA.*
- *instructions* Une ou plusieurs instructions VBA

Cette instruction VBA joue le même rôle que le IF... ELSEIF ... ENDIF précédent, mais est plus difficile à coder et à comprendre.
À éviter.

Autres fonctions de test à ne jamais utiliser:

```
Dim rRéponse
rRéponse = Switch(expression1,valeur1[, expression2, valeur2, expression3, valeur3, expression4, valeur4... [, expressionN, valeurN]])
```

Qui correspond à:

```
Dim rRéponse
If expression1 THEN
   rRéponse = valeur1
ELSEIF expression3 THEN
  rRéponse = valeur2
...
ELSE
   rRéponse = Null
ENDIF
```

```
Dim rRéponse
rRéponse = Choose(expression, valeur1[, valeur2, ... [, valeurN]])
```

Qui correspond à:

```
Dim rRéponse
If expression=1 THEN
   rRéponse = valeur1
ELSEIF expression=2 THEN
  rRéponse = valeur2
...
ELSE
   rRéponse = Null
ENDIF
```

Présentation des instructions de test

En VBA, il est convenu de présenter en retrait les instructions à l'intérieur d'un bloc de test.

Exemples à éviter:

```
If B > A Then A = B 'Ne pas faire de If sans Endif
```

```
If B > A Then
A = B 'Mettre les instructions d'un If en retrait
End If
```

Présentez ce test ainsi:

```
If B > A Then
   A = B 'Le retrait permet d'identifier les instructions
         'Dont l'exécution est conditionnelle au If
End If    'Le Endif permet d'identifier la fin du If
```

L'instruction de branchement

VBA hérite des anciennes versions de Basic une instruction de branchement, qui modifie la séquence d'exécution en effectuant un branchement vers une ligne identifiée par une étiquette.

La syntaxe de l'étiquette est:

nom:

Où *nom* respecte la syntaxe d'une variable VBA, et doit être suivi d'un deux points (:)

L'instruction
GoTo *étiquette*
peut être utilisée n'importe où à l'intérieur de la procédure (pas du module!) pour indiquer que la suite du traitement se trouve aux lignes suivant l'*étiquette*.

L'utilisation de **GoTo** n'est vraiment acceptable qu'à l'intérieur d'une instruction **On Error**.
Dans d'autres circonstances, elle rend la logique du programme difficile à comprendre, donc à modifier.

La fonction IIf()

VBA inclut la fonction IIf() qui fonctionne exactement comme la fonction Si() de Excel (français) ou If() (Excel anglais):

IIf(*expression***,** *ValeurVrai***,** *ValeurFaux***)**
Où:

- *expression* Expression à évaluer.
- *ValeurVrai* Valeur renvoyée si la valeur de *expression* est **True**.
- *ValeurFaux* Valeur renvoyée si la valeur de *expression* est **False**.

Fonctions de validation

VBA offre plusieurs séries de fonctions utiles pour valider la nature des données et des variables.

IsNumeric(*expression*)	Vrai si *expression* peut être évaluée comme une valeur numérique
IsDate(*expression*)	Vrai si *expression* peut être évaluée comme une date ou une heure

IsNull(*expression*)	Vrai si *expression* ne contient aucune variable de valeur **Null**. Attention: *expression* **= Null** *expression* **<> Null** sont TOUJOURS faux. Utilisez **IsNull**(Variable).
IsNothing(*variable*)	Vrai si *variableobjet* est un objet non initialisé (voir **SET**).
IsEmpty(*expression*)	Vrai si *expression* est de type **Variant** et n'est pas initialisée.
IsMissing(*paramètre*)	*paramètre* doit être le nom d'un paramètre facultatif de la procédure en cours. Faux si une valeur a été passée dans *paramètre*.

IsObject(*variable*)	Vrai si *variable* est de type **Object**. Utilisez plutôt la fonction **TypeName**()
IsArray(*variable*)	Vrai si *variable* est une variable contenant un tableau VBA.

TypeName(*variable*)	Retourne le nom du type de *variable*, exemples: • **Currency** • **Currency**() - tableau de type currency • **Date** • **Null** • **Object** • **Empty** mais aussi: • **Range** • **WorkSheet** • ...
VarType(*variable*)	Retourne une entier indiquant le type de *variable*. Beaucoup moins intéressant que **TypeName**().

Validation de paramètres

On a vu dans la section Déclarations qu'il est préférable de déclarer Variant les paramètres d'une Function ou d'un Sub paramétré.

En effet, il est impossible de prédire au moment de la programmation quel type de paramètre sera transmis à la Function ou au Sub.

Développons à une fonction fnNomMois qui retourne le nom du mois d'une date fournie en paramètre:	`Function fnNomMois(dDate As Date)`
	`'Auteur: Michel Berthiaume`
	`'Retourne le nom du mois de la date`
	`'fournie en paramètre`
	
	`    fnNomMois = Format(dDate, "mmmm")`
		Notez qu'on pourrait utiliser l'instruction IF et la fonction VBA Month() pour arriver au même résultat. Mais le code serait plus long et retournerait le nom de mois dans une seule langue.
		Le code à gauche utilise la fonction VBA Format pour obtenir le nom du mois de l'environnement Windows (donc dans la langue de l'utilisateur)

	End Function	
fnNomMois(#2012-1-2#)	retourne janvier	
fnNomMois(2012-1-2)	retourne juillet	Parce que 2012 moins 1 moins 2 vaut 2009 et que le numéro de jour 2009 correspond au 1er juillet 1905. VBA fait une conversion automatique non souhaitée.

Il est donc préférable de ne pas déclarer le type du paramètre et de plutôt en tester le type:	**Function** fnNomMois(dDate)
	'Auteur: Michel Berthiaume
	'Retourne le nom du mois de la date
	'fournie en paramètre
	
	If IsDate(dDate) **Then**
	  fnNomMois = **Format**(dDate, "mmmm")
	Else
	  fnNomMois = "Date invalide"
	End If
	
	End Function
		L'utilisateur sera alors informé qu'il a mal formulé le paramètre

Maintenant fnNomMois(2012-1-2)	retourne Date invalide	
Si la cellule Excel	et B1 contient =fnNomMois(A1)	
A1 contient 2012-01-02	B1 affiche janvier	
A1 ne contient rien	B1 affiche Date invalide	
A1 contient 2012-02-31	B1 affiche Date invalide	
A1 contient janvier	B1 affiche Date invalide	

Exemples

Exemple 1: Tester si une cellule Excel nommée "Nombre" contient un nombre.

```
Sub sExempleTest1()
'Auteur: Michel Berthiaume
'Vérifier si la cellule "Nombre" contient un nombre
    If IsNumeric(Range("Nombre").Value) Then
        MsgBox"La valeur est numérique"
    End If

    'Suite de la procédure
    '...

End Sub
```

Bien qu'intéressant, ce code n'est pas très fonctionnel.
Il affiche un message lorsque la cellule contient une valeur numérique (ou nulle),
mais ce qu'on veut probablement, c'est interrompre la suite de la procédure lorsque la valeur n'est pas numérique.

```
Sub sExempleTest2()
'Auteur: Michel Berthiaume
'Vérifier si la cellule "Nombre" contient un nombre
    If Not IsNumeric(Range("Nombre").Value) Then
        MsgBox"La valeur n'est pas numérique"
        Exit Sub
    End If
    'Suite de la procédure
    '...

End Sub
```

Exemple 2: Programmer une fonction qui retourne le nom d'une province, à partir d'un sigle.

```
Function fnNomProvince1(SigleProvince)
'Auteur: Michel Berthiaume
'Retourner le nom d'une province
Dim sSigleProvince As String

On Error GoTo Erreur:

    'D'abord tester la validité du paramètre
```

```
    If TypeName(SigleProvince) = "Range" Then
        sSigleProvince = UCase(SigleProvince.Value)
    Else
        If TypeName(SigleProvince) = "String" Then
            sSigleProvince = UCase(SigleProvince)
        Else
            fnNomProvince1 = "sigle inconnu"
            Exit Function
        End If
    End If

    'Rendu ici, sSigleProvince contient une version en majuscules du sigle
    If sSigleProvince = "AB" Then
        fnNomProvince1 = "Alberta"
    Else
        If sSigleProvince = "BC" Then
            fnNomProvince1 = "Colombie Britannique"
        Else
            If sSigleProvince = "PE" Then
                fnNomProvince1 = "Île-du-Prince-Édouard"
            Else
                If sSigleProvince = "NS" Then
                    fnNomProvince1 = "Nouvelle-Écosse"
                Else
                    If sSigleProvince = "NU" Then
                        fnNomProvince1 = "Nunavut"
                    Else
                        If sSigleProvince = "ON" Then
                            fnNomProvince1 = "Ontario"
                        Else
                            If sSigleProvince = "QC" Then
                                fnNomProvince1 = "Québec"
                            Else
                                If sSigleProvince = "SK" Then
                                    fnNomProvince1 = "Saskatchewan"
                                Else
                                    If sSigleProvince = "NF" Then
                                        fnNomProvince1 = "Terre Neuve"
                                    Else
                                        If sSigleProvince = "NT" Then
                                            fnNomProvince1 = "Territoires du Nord-Ouest"
                                        Else
                                            If sSigleProvince = "YK" Then
                                                fnNomProvince1 = "Yukon"
                                            Else
                                                fnNomProvince1 = "Sigle inconnu"
                                            End If
                                        End If
                                    End If
                                End If
                            End If
                        End If
                    End If
                End If
            End If
        End If
    End If

    Exit Function
Erreur:
    fnNomProvince1 = "Erreur: " & Err.Description
End Function
```

Bien que fonctionnel, le code ci-dessus est lourd et répétitif.
Voici une version allégée:

```
Function fnNomProvince2(SigleProvince)
'Auteur: Michel Berthiaume
'Retourner le nom d'une province
Dim sSigleProvince As String

On Error GoTo Erreur:

    'D'abord tester la validité du paramètre
    If TypeName(SigleProvince) = "Range" Then
        sSigleProvince = UCase(SigleProvince.Value)
    Else
        If TypeName(SigleProvince) = "String" Then
            sSigleProvince = UCase(SigleProvince)
        Else
```

```
            fnNomProvince2 = "sigle inconnu"
            Exit Function
        End If
    End If

    'Rendu ici, sSigleProvince contient une version en majuscules du sigle
    If sSigleProvince = "AB" Then
        fnNomProvince2 = "Alberta"
    ElseIf sSigleProvince = "BC" Then
        fnNomProvince2 = "Colombie Britannique"
    ElseIf sSigleProvince = "PE" Then
        fnNomProvince2 = "Île-du-Prince-Édouard"
    ElseIf sSigleProvince = "NS" Then
        fnNomProvince2 = "Nouvelle-Écosse"
    ElseIf sSigleProvince = "NU" Then
        fnNomProvince2 = "Nunavut"
    ElseIf sSigleProvince = "ON" Then
        fnNomProvince2 = "Ontario"
    ElseIf sSigleProvince = "QC" Then
        fnNomProvince2 = "Québec"
    ElseIf sSigleProvince = "SK" Then
        fnNomProvince2 = "Saskatchewan"
    ElseIf sSigleProvince = "NF" Then
        fnNomProvince2 = "Terre Neuve"
    ElseIf sSigleProvince = "NT" Then
        fnNomProvince2 = "Territoires du Nord-Ouest"
    ElseIf sSigleProvince = "YK" Then
        fnNomProvince2 = "Yukon"
    Else
        fnNomProvince2 = "Sigle inconnu"
    End If

    Exit Function
Erreur:
    fnNomProvince2 = "Erreur: " & Err.Description
End Function
```

Il est aussi possible d'utiliser l'Instruction SELECT, mais elle n'offre pas d'avantage important sur la version précédente:

```
Function fnNomProvince3(SigleProvince)
'Auteur: Michel Berthiaume
'Retourner le nom d'une province
Dim sSigleProvince As String

On Error GoTo Erreur:

    'D'abord tester la validité du paramètre
    If TypeName(SigleProvince) = "Range" Then
        sSigleProvince = UCase(SigleProvince.Value)
    Else
        If TypeName(SigleProvince) = "String" Then
            sSigleProvince = UCase(SigleProvince)
        Else
            fnNomProvince3 = "sigle inconnu"
            Exit Function
        End If
    End If

 'Rendu ici, sSigleProvince contient une version en majuscules du sigle
    Select Case sSigleProvince
        Case "AB"
            fnNomProvince3 = "Alberta"
        Case "BC"
            fnNomProvince3 = "Colombie Britannique"
        Case "PE"
            fnNomProvince3 = "Île-du-Prince-Édouard"
        Case "NS"
            fnNomProvince3 = "Nouvelle-Écosse"
        Case "NU"
            fnNomProvince3 = "Nunavut"
        Case "ON"
            fnNomProvince3 = "Ontario"
        Case "QC"
            fnNomProvince3 = "Québec"
        Case "SK"
            fnNomProvince3 = "Saskatchewan"
        Case "NF"
            fnNomProvince3 = "Terre Neuve"
        Case "NT"
```

```
            fnNomProvince3 = "Territoires du Nord-Ouest"
        Case "YK"
            fnNomProvince3 = "Yukon"
        Case Else
            fnNomProvince3 = "Sigle inconnu"
    End Select

    Exit Function
Erreur:
    fnNomProvince3 = "Erreur: " & Err.Description
End Function
```

Sommaire

VBA offre plusieurs structures contrôlant l'ordre d'exécution des instructions.

- La structure de branchement (GoTo) ne doit être utilisée que pour l'établissement de la gestion des erreurs (On Error GoTo).
- Les structures de décision. La plus utilisée (et celle qui est recommandée) est
  If ... Then/.../ElseIf ... Then/.../Else/.../Endif
- Les structures de boucle, décrites dans la prochaine section.
- La présentation en retrait des lignes composant une structure de décision est esssentielle à sa documentation, comme les parenthèses dans une expression.

Fichier exemples

Le fichier Excel Exemples Tests.xlsm contient les exemples présentés dans la présente section.

Suite: Boucles

VBA Excel

par Michel Berthiaume

http://www.usherbrooke.ca/adm/departements/simqg/professeurs/michel-berthiaume/

Boucles

Sur cette page...

Les prochaines pages...

Les instructions de boucle

Les instructions d'un programme sont normalement exécutées les unes à la suite des autres, séquentiellement. Les instructions de boucle, de test et de branchement permettent de modifier cet ordre naturel.

On utilise les instructions de boucle pour que le programme exécute un bloc d'instructions plusieurs fois.

For Each *objet* **In** *collection*
    [*instructions*]
[**Exit For**]
    [*instructions*]
Next [*objet*]

Où:

- *objet* Variable utilisée pour être répétée sur tous les éléments d'une collection ou d'un tableau. Dans le cas d'une collection, la variable *objet* peut uniquement être une variable de type Variant, une variable objet générique ou toute variable objet spécifique. Dans le cas d'un tableau, il peut uniquement s'agir d'une variable de type Variant.
- *collection* Nom d'une collection ou d'un tableau d'objets (à l'exception d'un tableau de type défini par l'utilisateur).
- *instructions* Une ou plusieurs instructions exécutées pour chaque élément de la *collection*.

Le bloc d'*instructions* est exécuté autant de fois qu'il y a d'objets dans la collection. À chaque itération, l'objet suivant de la collection *collection* est assigné à la variable *objet.*

La boucle est terminée lorsque chaque objet a été traité, ou lorsque l'instruction **Exit For** est exécutée. Le programme se poursuit à l'instruction suivant le **Next.**

Cette première version de l'instruction **For** ... **Next** est la plus utilisée pour manipuler les plages de cellules Excel. Elle permet de traiter simplement toutes les cellules d'une plage, sans se soucier de leur adresse absolue ou relative.
Elle manipule aussi, mais plus difficilement, les **tableaux VBA**.

Une plus ancienne version de l'instruction **For** ... **Next** est plus générale, mais requiert l'utilisation d'un compteur:

For *compteur* = *début* **To** *fin*[**Step** *valeur*]
  [*instructions*]
  [**Exit For**]
  [*instructions*]
Next [*compteur*]

Où:

- *compteur* Variable numérique utilisée comme un compteur.
- *début* Valeur assignée au compteur lors de la première exécution des *instructions.*
- *fin* Valeur qui détermine la fin de la boucle.
- *valeur* Valeur ajoutée au *compteur* à chaque exécution de la boucle.
  1 par défaut.
- *instructions* Une ou plusieurs instructions entre **For** et **Next**.

En principe, les instructions entre **For** et **Next** sont exécutées (*fin - début + 1*) fois, en ajoutant *valeur* à *compteur* à chaque itération. Bien noter que les bornes *fin* et *début* sont inclues.
L'instruction **Exit For** peut être utilisée pour une sortie exceptionnelle de la boucle.

À la fin de l'exécution de la boucle (après le **Next**), *compteur* vaut *fin* + *valeur*, et non *fin.*

DANGER: ne jamais assigner de valeur à *compteur* à l'intérieur des *instructions*.

Cette 2e version de **For...Next** est plus ancienne que la précédente, et est plus complexe à utiliser pour des collections d'objet.
Par contre, elle est plus adaptée à la manipulation de **tableaux VBA**.
Elle correspond au code VBA suivant:

compteur = début
Do Until *compteur > fin*
  [*instructions*]
  [**Exit Do**]
  [*instructions*]
  compteur = compteur + valeur
Loop

Ce qui nous amène aux instructions de répétition plus générales:

Do While *condition*
  [*instructions*]
  [**Exit Do**]
  [*instructions*]
Loop

Do Until *condition*
  [*instructions*]
  [**Exit Do**]
  [*instructions*]
Loop

Do
  [*instructions*]
  [**Exit Do**]
  [*instructions*]
Loop While *condition*

Do
  [*instructions*]
  [**Exit Do**]
  [*instructions*]
Loop Until *condition*

Où

- *condition* Expression VBA. Si elle vaut Null, elle est considérée Faux.
- *instructions* Une ou plusieurs instructions répétées tant que *condition* est Vrai (**While**), ou jusqu'à ce qu'elle le devienne (**Until**).

Ces quatre instructions implantent en VBA les structures de boucle structurées. Elles sont peu utilisées, sauf pour les boucles de lecture.

Présentation des instructions de boucle

En VBA, il est convenu de présenter en retrait les instructions à l'intérieur d'une boucle.

Exemples à éviter:

- For Each Cellule in Plage:MsgBox Cellule.Value:Next 'Ne pas mettre plusieurs instructions VBA sur la même ligne
- For Each Cellule in Plage
  MsgBox Cellule.Value 'Mettre les instructions répétées en retrait
  Next

Présentez cette boucle ainsi:

```
For Each Cellule in Plage
   MsgBox Cellule.Value     'Mettre les instructions répétées en retrait
Next
```

Exemple 1: boucle de collection

Supposons une plage de cellules Excel nommée ListeProvinces:

On peut écrire une fonction VBA qui retourne le nom associé au sigle:

```
Function fnNomProvince4(SigleProvince)
'Auteur: Michel Berthiaume
'Retourner le nom d'une province

Dim sSigleProvince As String
Dim rCellule As Range

On Error GoTo erreur: 'au cas où

    fnNomProvince4 = "sigle inconnu" 'Réponse par défaut

    'D'abord tester la validité du paramètre
    If TypeName(SigleProvince) = "Range" Then
        sSigleProvince = UCase (SigleProvince.Value)
    ElseIf TypeName(SigleProvince) = "String" Then
        sSigleProvince = UCase(SigleProvince)
    Else
        Exit Function
    End If

    'Rendu ici, sSigleProvince contient une version en majuscules du sigle
    'Pour chaque cellule de colonne 1 de la plage nommée ListeProvinces
    For Each rCellule In Range("ListeProvinces").Columns(1).Cells
        If sSigleProvince = UCase(rCellule.Value) Then 'Valeur trouvée
            fnNomProvince4 = rCellule.Offset(0, 1).Value 'Valeur de la cellule à droite
            Exit For 'inutile de continuer: on sort de la boucle.
        End If
    Next

    Exit Function 'Pour éviter l'exécution des instructions de gestion d'erreur

erreur:
    fnNomProvince4 = "Erreur inattendue, contacter le programmeur." & Err.Description

End Function
```

La même, en utilisant un compteur de lignes:

```
Function fnNomProvince5(SigleProvince)
'Auteur: Michel Berthiaume
'Retourner le nom d'une province

Dim sSigleProvince As String
```

```
Dim lLigne as Long

On Error GoTo erreur: 'au cas où
    fnNomProvince5 = "sigle inconnu" 'Réponse par défaut

    'D'abord tester la validité du paramètre
    If TypeName(SigleProvince) = "Range" Then
        sSigleProvince = UCase (SigleProvince.Value)
    ElseIf TypeName(SigleProvince) = "String" Then
        sSigleProvince = UCase(SigleProvince)
    Else
        Exit Function
    End If

    'Rendu ici, sSigleProvince contient une version en majuscules du sigle
    'On fait varier lLigne de 1 au nombre de lignes de la plage nommée ListeProvinces
    For lLigne = 1 to Range("ListeProvinces").Rows.Count
        If sSigleProvince = Range("ListeProvinces").Cells(lLigne, 1).Value Then
            fnNomProvince5 = Range("ListeProvinces").Cells(lLigne, 2).Value 'Valeur de la cellule à droite
            Exit For 'inutile de continuer: on sort de la boucle.
        End If
    Next


    Exit Function 'Pour éviter l'exécution des instructions de gestion d'erreur

erreur:
    fnNomProvince5 = "Erreur inattendue, contacter le programmeur." & Err.Description

End Function
```

Exemple 2: boucle avec compteur et accumulateur

On veut une fonction fnMoyennePlus qui calcule la moyenne des valeurs positives d'une plage.

```
Function fnMoyennePlus(Plage)
'Auteur: Michel Berthiaume
'Calculer la moyenne des valeurs positives d'une plage

Dim cSomme as Currency
Dim cCompteur as Currency
Dim rCellule as Range

On Error GoTo erreur: 'au cas où
    'D'abord tester la validité du paramètre
    If TypeName(Plage) <> "Range" Then
        Exit Function
    End If

    For each rCellule in Plage
        If rCellule.Value > 0 Then
            cSomme = cSomme + rCellule.Value
            cCompteur = cCompteur +1
        End If
    Next
    If cCompteur > 0 Then    'Éviter les divisions par 0
        fnMOyennePlus = cSomme/cCompteur
    EndIf


    Exit Function 'Pour éviter l'exécution des instructions de gestion d'erreur

erreur:
    fnMoyennePlus = "Erreur inattendue, contacter le programmeur." & Err.Description

End Function
```

Une variante de la même fonction:

```
Function fnMoyennePlus1(Plage)
'Auteur: Michel Berthiaume
'Calculer la moyenne des valeurs positives d'une plage

Dim lNoCellule As Long
Dim cSomme as Currency
Dim cCompteur as Currency
```

```
Dim rCellule as Range

On Error GoTo erreur: 'au cas où
    'D'abord tester la validité du paramètre
    If TypeName(Plage) <> "Range" Then
        Exit Function
    End If

    For lNoCellule = 1 to Plage.Cells.Count
        If Plage.Cells(lNoCellule).Value > 0 Then
            cSomme = cSomme + Plage.Cells(lNoCellule).Value
            cCompteur = cCompteur + 1
        End If
    Next
    If cCompteur > 0 Then    'Éviter les divisions par 0
        fnMoyennePlus1 = cSomme/cCompteur
    EndIf


    Exit Function 'Pour éviter l'exécution des instructions de gestion d'erreur

erreur:
    fnMoyennePlus1 = "Erreur inattendue, contacter le programmeur." & Err.Description

End Function
```

Finalement, une variante plus classique de cette solution, utilisant un compteur de lignes et un compteur de colonnes:

```
Function fnMoyennePlus2(Plage)
'Auteur: Michel Berthiaume
'Calculer la moyenne des valeurs positives d'une plage

Dim lNoLigne As Long
Dim lNoColonne As Long
Dim cSomme as Currency
Dim cCompteur as Currency
Dim rCellule as Range


On Error GoTo erreur: 'au cas où
    'D'abord tester la validité du paramètre
    If TypeName(Plage) <> "Range" Then
        Exit Function
    End If

    For lNoLigne = 1 to Plage.Rows.Count              'Pour chaque ligne
        For lNoColonne = 1 to Plage.Columns.Count     'Pour chaque colonne
            If Plage.Cells(lNoLigne,lNoColonne).Value > 0 Then
                cSomme = cSomme + Plage.Cells(lNoLigne,lNoColonne).Value
                cCompteur = cCompteur + 1
            End If
        Next
    Next
    If cCompteur > 0 Then    'Éviter les divisions par 0
        fnMoyennePlus1 = cSomme/cCompteur
    EndIf
    Exit Function 'Pour éviter l'exécution des instructions de gestion d'erreur

erreur:
    fnMoyennePlus2 = "Erreur inattendue, contacter le programmeur." & Err.Description
End Function
```

Exemple 3: boucle de lecture

Dans une boucle de lecture, le nombre d'itérations n'est pas connu au départ. L'utilisation de For...Next n'est donc pas approprié.

Ici, on veut saisir une série de nombres qui seront inscrits dans la feuille Excel.

```
Sub SaisirNombres()
'Auteur: Michel Berthiaume
'Saisir des nombres et les inscrire dans une feuille Excel

Dim sSaisie As String
Dim lNoLigne As Long
```

```
On Error GoTo erreur: 'au cas où

    'Dans la boucle de lecture "standard", la première valeur est lue AVANT d'entrer dans la boucle
    sSaisie = InputBox("Entrer une valeur", "Inscription de valeurs dans une feuille")

    Do Until sSaisie = ""             'Le bouton Annuler de InputBox renvoie la chaîne nulle
        If IsNumeric(sSaisie) Then
            lNoLigne = lNoLigne + 1   'Prochaine ligne de la feuille Excel
            Range("A1").Offset(lNoLigne, 0).Value = sSaisie
        End If
        sSaisie = InputBox("Entrer une valeur", "Inscription de valeurs dans une feuille")
    Loop

    Exit Sub 'Pour éviter l'exécution des instructions de gestion d'erreur

erreur:
    MsgBox "Erreur inattendue, contacter le programmeur." & Err.Description

End Sub
```

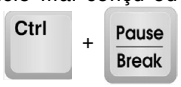
Ce type de boucle est utilisé pour lire des données venant de requêtes SQL, de fichiers conventionnels, etc.
Notez qu'il y a lecture AVANT d'entrer dans la boucle, puis une autre immédiatement avant la fin, et que le test de fin de lecture est en début de boucle.
La première instruction de lecture est automatique (implicite) dans l'ouverture d'une requête SQL.

Une autre façon d'arriver au même résultat:

```
Sub SaisirNombres1()
'Auteur: Michel Berthiaume
'Saisir des nombres et les inscrire dans une feuille Excel

Dim sSaisie As String
Dim lNoLigne As Long

On Error GoTo erreur: 'au cas où
    Do
        Do
            sSaisie = InputBox("Entrer une valeur", "Inscription de valeurs dans une feuille")
        Loop Until IsNumeric(sSaisie) Or Len(sSaisie) = 0
        If Len(sSaisie) = 0 Then
            lNoLigne = lNoLigne + 1   'Prochaine ligne de la feuille Excel
            Range("A1").Offset(lNoLigne, 0).Value = sSaisie
        End If
    Loop Until sSaisie = ""             'Le bouton Annuler de InputBox renvoie la chaîne nulle

    Exit Sub 'Pour éviter l'exécution des instructions de gestion d'erreur

erreur:
    MsgBox "Erreur inattendue, contacter le programmeur." & Err.Description

End Sub
```

Sommaire

VBA offre plusieurs structures contrôlant l'ordre d'exécution des instructions.

- Les structures de décision vues dans la section précédente.
- Les structures de boucle ou répétition:
  - For Each ... Next qui traite tour à tour chaque élément d'une collection.
  - For compteur= n To m ... Next qui fait varier compteur de n à m.
  - Do ... Loop qui implante en VBA les 4 structures de boucle classique.
- La présentation en retrait des lignes à répéter à l'intérieur de la boucle est esssentielle à sa documentation, comme les parenthèses dans une expression.
- Un test de fin de boucle mal conçu ou omis peut provoquer une boucle sans fin lors de l'exécution. L'utilisateur a alors l'impression que le programme est "gelé". Si ça se produit lors de tests, utiliser 

pour interrompre l'exécution. Si votre clavier ne comporte pas de touche Pause/Break, contactez le fabricant de l'ordinateur pour trouver la combinaison de touches équivalente ou utilisez un autre clavier. Cette touche est indispensable pour déboguer les programmes.

Fichier exemples

Le fichier Excel Tutoriel Exemples Boucles.xlsm contient les exemples présentés dans la présente section.

Suite: Gestion d'erreur

VBA Excel

par Michel Berthiaume

http://www.usherbrooke.ca/adm/departements/simqg/professeurs/michel-berthiaume/

Gestion d'erreur

Sur cette page...

Les prochaines pages...

Sortes d'erreur

- **erreurs de syntaxe**: une instruction ne respectant pas la syntaxe VBA:

`Sub test`
`    Dim sNom as Sting`
`End Sub`
	Message:
	Erreur de compilation:
	Type défini par l'utilisateur non défini
	Cause:
	Sting n'est pas un type valide

`Sub Test`
`    MsgBox Bonjour`
`End Sub`
	Message:
	Erreur de compilation:
	Variable non définie
	Cause:
	Il manque les "

`Sub Test`
`Dim i as long`
`    For i = 1 to 20`
`End Sub`
	Message:
	Erreur de compilation:
	For sans Next

`Sub Test`
`    If Not isNumeric(InputBox("Entrer une valeur")) then`
`        MsgBox "Ce n'est pas une valeur"`
`End Sub`
	Message:
	Erreur de compilation:
	Bloc If sans End If

`Sub Test`
`Dim sRéponse as String`
`    Do`
`        Réponse = InputBox("Entrer une valeur")`
`    Loop until not isNumeric(sRéponse)`
`End Sub`
	Message:
	Erreur de compilation:
	Variable non définie
	
	Cause:
	sRéponse, pas Réponse, dans le Do

Ces erreurs sont détectées par VBE dès qu'on tente de passer à la ligne suivante, ou à chaque fois qu'on tente d'exécuter la procédure.
Elles sont causées par un manque d'attention ou de connaissance du langage.

- **erreurs d'exécution prévisibles:** une suite d'instructions ne respectant pas la logique VBA.

`Sub Test`
`Dim cRéponse as Currency`
`    Do`
`        cRéponse =InputBox("Entrer une valeur")`
`    Loop until not isNumeric(cRéponse)`
`End Sub`
	Si l'utilisateur fournit une réponse non numérique:
	
	erreur d'exécution 13:
	Incompatibilité de type

`Function fnMoyenneMoinsMin(rPlage)`
`'Calculer la moyenne`
`'moins la valeur la plus basse des valeurs d'une plage`
`Dim cSomme As Currency`
`Dim cMin As Currency`
`Dim rCellule As Range`
 	
`    'Initialiser à la 1ère valeur de la plage`
`    cMin = rPlage.Cells(1, 1).Value`
 	
`    For Each rCellule In rPlage`
`        cSomme = cSomme + rCellule.Value`
`        cMin = fnMin(rCellule.Value, cMin)`
`    Next`
 	
`    fnMoyenneMoinsMin = (cSomme - cMin) / (rPlage.Count - 1)`
`End Function`
	Si le paramètre n'est pas une plage:
	Message:
	erreur d'exécution 424:
	Objet requis
	
	Si rPlage ne comporte qu'une cellule, on a une division par 0:
	
	Message:
	erreur d'exécution 6:
	Dépassement de capacité

`Function fnMin(a, b)`
`'Retourne le minimum de a ou b`
`    If a < b Then`
`        fnMin = a`
`    Else`
`        fnMin = b`
`    End If`
`End Function`
	Si a et b sont de nature différente (comparer 3 et Nothing, par exemple):
	
	Message:
	erreur d'exécution 13:
	Incompatibilité de type

Ces erreurs ne sont détectées par VBA que lorsqu'on exécute la procédure dans certaines circonstances, des conditions limites.
Elles sont causées par des maladresses de programmeurs qui oublient qu'une procédure peut être utilisée dans des contextes différents de ceux où elle est développée.

- **erreurs d'exécution imprévisibles:** une suite d'instructions qui ne fonctionnent pas dans des conditions inattendues.

`Sub EnregistrerClasseur`
`    ActiveWorkbook.Save`
`End Function`
	Si le lecteur (dossier) actif est en lecture seule:
	Message:
	erreur d'exécution 1004:
	Impossible d'accéder au document en lecture seule

Ces erreurs ne sont détectées par VBA que lorsque les circonstances imprévues se produisent.
Elles sont causées par l'utilisation d'un programme en dehors des contextes prévisibles.

Si les messages d'erreur affichés par VBE (erreurs de compilation) sont assez faciles à gérer, puisqu'ils apparaissent avant même qu'on termine ou utilise la procédure, les messages d'erreur VBA (erreurs d'exécution) sont affichés lors de l'exécution, ce qui n'est acceptable qu'en phase de test.
Les messages d'erreur d'exécution sont habituellement incompréhensibles par les utilisateurs ne connaissant pas VBA.
Et souvent par ceux qui le connaissent.

Les instructions de gestion d'erreur

On Error GoTo *étiquette*	Active un branchement différé. Si (lorsque) une erreur d'exécution se produit, l'exécution se poursuit à partir de la ligne identifiée par l'*étiquette.* Ce branchement différé demeure en vigueur jusqu'à l'exécution d'une autre instruction **On Error** ou à la fin de la procédure **Sub** ou **Function**.
On Error Resume Next	Si (lorsque) une erreur d'exécution se produit, VBA passe à l'instruction suivant celle ayant causé l'erreur.
On Error GoTo 0	Désactive toute gestion d'erreur par une ou l'autre forme de **On Error**.
Resume	Valide seulement lorsqu'une erreur a été détectée par un branchement différé **On Error**. L'exécution reprend à partir de l'instruction qui a provoqué l'erreur.
Resume Next	Valide seulement lorsqu'une erreur a été détectée par un branchement différé **On Error**. L'exécution reprend à partir de l'instruction suivant celle qui a provoqué l'erreur.

Resume *étiquette*	Valide seulement lorsqu'une erreur a été détectée par un branchement différé **On Error**. L'exécution reprend à la ligne indiquée par l'*étiquette*.

Err

Objet dont les propriétés sont permettent d'identifier une erreur d'exécution. Les propriétés sont initialisées lorsqu'un branchement différé activé par **On Error** est exécuté.

Principales propriétés:

- **Number** Contient le numéro de l'erreur.
- **Description** Contient la description de l'erreur.

Les autres particularités de l'objet **Err** ne sont intéressantes que pour des applications très avancées.

Exemple: Capturer n'importe quelle erreur

```
Function fnMoyenneMoinsMin(rPlage)
'Auteur: Michel Berthiaume
'Calculer la moyenne
'moins la valeur la plus basse des valeurs d'une plage
Dim cSomme As Currency
Dim cMin As Currency
Dim rCellule As Range

    On Error GOTO Erreur:

    'Initialiser à la 1ère valeur de la plage
    cMin = rPlage.Cells(1, 1).Value

    For Each rCellule In rPlage
        cSomme = cSomme + rCellule.Value
        cMin = fnMin(rCellule.Value, cMin)'Noter l'utilisation d'une fonction dans une fonction
    Next

    fnMoyenneMoinsMin = (cSomme - cMin) / (rPlage.Count - 1)

    Exit Function

Erreur:
    fnMoyenneMoinsMin = "Erreur: " & Err.Description

End Function
```

Remarquez la présence de **Exit Function** avant l'étiquette Erreur: qui évite que la gestion d'erreur soit exécutée lorsqu'il n'y a pas d'erreur.
La présence d'une étiquette n'interrompt pas la procédure.

Exemple: Capturer une erreur spécifique

```
Sub EnregistrerClasseur()
'Auteur: Michel Berthiaume
'enregistrer le classeur

    On Error GoTo Erreur:

    ActiveWorkbook.Save

    Exit Sub

Erreur:
    If Err.Number = 1004 Then
        If MsgBox("Le fichier est en lecture seule. SVP corriger et cliquer Ok" & _
            vbCrLf & "ou cliquer annuler", vbOKCancel) = vbOK Then
            Resume
        End If
    End If
End Sub
```

Remarquez la présence de **Exit Sub** avant l'étiquette Erreur: qui évite que la gestion d'erreur soit exécutée lorsqu'il n'y a pas d'erreur.

La présence d'une étiquette n'interrompt pas la procédure.

Exemple: utiliser la gestion d'erreur à d'autres fins

```
Function fnMin(a, b)
'Auteur: Michel Berthiaume
'Retourne le minimum de a ou b
'Retourne "" si la comparaison est impossible

On Error GoTo Erreur

    If a < b Then
        fnMin = a
    Else
        fnMin = b
    End If

    Exit Function

Erreur: 'Gère les comparaisons impossibles, comme les pommes vs les oranges
    fnMin = ""
End Function
```

Remarquez la présence de **Exit Function** avant l'étiquette Erreur: qui évite que la gestion d'erreur soit exécutée lorsqu'il n'y a pas d'erreur.
La présence d'une étiquette n'interrompt pas la procédure.

Sommaire

Tout programme doit prévoir les erreurs (exceptions) prévisibles et imprévisibles.

On utilise les instructions de test pour gérer les erreurs prévisibles (validations).

On utilise les instructions de capture et gestion d'erreurs (On Error GoTo) pour spécifier les instructions à exécuter si une erreur se produit lors de l'exécution du programme.

Il faut éviter d'utiliser On Error Resume Next qui permet au programme d'ignorer l'instruction ayant généré l'erreur.

La structure générale d'un programme VBA est donc la suivante:

```
Sub ou Function...
'Auteur:...
'Description: ...
Dim ... 'Toutes les déclarations

On Error GoTo Erreur:    'Installer la capture d'erreurs

'Validations: prévoir les exceptions
      'Exit Sub ou exit Function    'Pour terminer le programme au besoin
...

'Traitement normal
...

Exit Sub ' Ou exit Function    'Fin normale du programme

Erreur:
...       'Traitement des erreurs imprévisibles, souvent un message à l'utilisateur

End Sub 'Ou End Function
```

Fichier exemples

Le fichier Excel Tutoriel Exemples Erreurs.xlsm contient les exemples présentés dans la présente section.

Suite: Collections et tableaux

VBA Excel

par Michel Berthiaume

http://www.usherbrooke.ca/adm/departements/simqg/professeurs/michel-berthiaume/

Collections et tableaux

Sur cette page...

Les prochaines pages...

Collections

Une collection est un groupe d'instances d'une même classe (Wikipedia).

Exemples:

- un classeur Excel contient une collection de feuilles Excel
- une feuille Excel contient une collection de plages Excel
- une plage Excel contient une collection ... de plages Excel!

La propriété **Count** contient le nombre d'objets de la collection.

On peut manipuler les objets d'une collection de diverses façons:

- par son numéro:
  Sheets(1) est la 1ère feuille du classeur actif
- par son nom:
  Sheets("Provinces") est la feuille Provinces du classeur actif.
- en utilisant l'instruction **For Each**:

```
Dim sFeuille As Worksheet
For Each sFeuille In Worksheets
    ...
Next
```

qui est l'équivalent de:

```
Dim sFeuille As Worksheet, i As Long
For i = 1 To Worksheets.Count
    Set sFeuille = Worksheets(i)
Next
```

Tableaux

On peut concevoir un tableau VBA comme une collection de variables VBA, mais comme les tableaux existaient en Basic bien avant l'introduction de la programmation objet, le traitement des tableaux diffère du traitement des collections.

Un tableau VBA doit être déclaré.

Le nombre d'éléments d'un tableau peut être fixé lors de la déclaration (nombre fixe d'éléments) ou lors de l'exécution (tableau dynamique).

Exemples:

```
Dim tTableau(1 to 15) As Currency         '15 éléments de type Currency
Dim tTableau(15) As Currency              '16 éléments de type Currency
Dim tTableau() As Currency
...
Redim tTableau(1 to 15)                   '15 éléments de type Currency
Dim tTableau(1 To 5, 1, To 5) As Currency '25 éléments de type Currency
```

La seule façon d'utiliser un élément d'un tableau est d'utiliser son adresse:

```
tTableau(1)      'Le premier élément du tableau
tTableau(10)     'Le 10e élément du tableau
tTableau(1, 5)   'Le 1er élément de la 5e colonne
```

Lorsqu'un tableau a une dimension, comme tTableau(1 to 15), c'est un vecteur, et on le représente comme une liste.

Lorsqu'un tableau a deux dimensions, comme tTableau(1 to 5, 1 To 5), c'est une matrice, et on le représente comme un tableau, le premier indice étant le numéro de ligne, le second le numéro de colonne.

Lorsqu'un tableau a trois dimensions, comme tTableau(1 to 5, 1 To 5, 1 To 5), c'est un cube, et on ne le représente pas.
Le premier indice est le numéro de ligne, le second le numéro de colonne et le troisième le numéro de rangée (profondeur).

Un tableau peut avoir plus de 3 dimensions, mais les applications sont rares.

Instructions de tableaux

Certaines instructions VBA sont spécifiques aux tableaux:

ReDim [**Preserve**] *nom*(*indices*) [**As** *type*] [, *nom*(*indices*) [**As** *type*]] **. . .**

Où:

- **Preserve** Indique que les données existante avant de redimensionnement doivent être conservées. Rarement utilisé.
- *nom* Nom de la variable.
- *indices* Dimensions du tableau Utilisez la forme:
  [*Min* **To**] *max*[,[*min* **To**] *max*] ...
  Où *min* est la valeur plancher de l'indice et *max* la valeur plafond
  Valeur par défaut de min: 0 et de max: 10
- *type* Type de données de la variable.

Utilisez **ReDim** pour les tableaux dont les dimensions ne sont pas précisées dans l'instruction Dim.

LBound(*NomTableau*[, *dimension*]**)**

Où

- *NomTableau* Est le nom d'un tableau VBA.
- *dimension* Est le numéro de la dimension dont LBound retournera la valeur minimale. Défaut 1.

LBound est utilisé pour obtenir la valeur minimale (Lower Bound) de la *dimension* d'un tableau VBA.

UBound(*NomTableau*[, *dimension*]**)**

Où

- *NomTableau* Est le nom d'un tableau VBA.
- *dimension* Est le numéro de la dimension dont UBound retournera la valeur maximale. Défaut 1.

UBound est utilisé pour obtenir la valeur maximale (Upper Bound) de la *dimension* d'un tableau VBA.

Ces trois instructions sont utiles lorsqu'on utilise un tableau dynamique, c'est à dire dont la dimension n'est connue qu'au moment de l'exécution (comme une liste d'étudiants ou d'items d'une facture), par opposition à un tableau statique, dont le nombre d'éléments est connu au moment de l'écriture du programme (une liste de noms de mois, par exemple).

Plage ou tableau?

Si on a une liste de valeurs à manipuler ou produire en VBA Excel, doit-on le faire en utilisant un plage Excel ou un tableau VBA?

Avantages de la plage Excel:

- plus facile à déboguer car les valeurs sont visibles dans Excel
- on peut utiliser les fonctions Excel

Avantages du tableau VBA:

- beaucoup plus rapide si on a un nombre considérable d'éléments
- le code est plus facilement réutilisable en Access, Word, VB6 ou même VB.net.

Copier des données entre Excel et VBA

Vous pouvez copier directement une plage Excel dans un tableau VBA dynamique de type Variant, sans même utiliser l'instruction **ReDim**:

```
Dim vTableau() As Variant
vTableau = Range("Plage")
```

a l'effet suivant:

- vTableau est redimensionné au nombre de lignes et de colonnes de la plage Plage
- le contenu de plage est copié dans vTableau

Bien noter que vTableau doit être un tableau dynamique de type Variant.

Vous pouvez copier directement un tableau VBA dans une plage Excel:

```
Range(Cells(1, 1), Cells(UBound(vTableau, 1), UBound(vTableau, 2))) = vTableau
```

copie tout le tableau vTableau dans la plage de même dimension commençant en A1 de la feuille active.
vTableau peut être de n'importe quel type et sera converti dans le type Excel correspondant.

Exemples

```
Sub ExemplesTableaux()
'Auteur: Michel Berthiaume
'Exemples de manipulation de collection et de tableau
Dim cNombres() As Currency
Dim vTableau() As Variant
Dim lLigne As Long, lColonne As Long
Dim tDépart As Date

On Error GoTo Erreur:

With Worksheets("Tableaux")

    Cells.Clear

    'Créer une matrice 10000x100 de nombres dans Excel
    tDépart = Now 'Date et heure du départ

    For lLigne = 1 To 2000
        Application.StatusBar = "Traitement de la ligne Excel " & lLigne & " de 2000"
        For lColonne = 1 To 100
            .Cells(lLigne, lColonne) = lLigne + lColonne
        Next
```

```
    Next
    MsgBox "Création de matrice Excel en " & Round((Now - tDépart) * 24 * 60 * 60, 2) _
        & " secondes"

    'Créer une matrice 2000x100 de nombres dans VBA
    ReDim cNombres(1 To 2000, 1 To 100)
    tDépart = Now 'Date et heure du départ
    For lLigne = 1 To 2000
        Application.StatusBar = "Traitement de la ligne de tableau VBA " & lLigne & " de 2000"
        For lColonne = 1 To 100
            cNombres(lLigne, lColonne) = lLigne + lColonne
        Next
    Next
    MsgBox "Création de matrice VBA en " & Round((Now - tDépart) * 24 * 60 * 60, 10) & " secondes"

    'Copie d'une matrice Excel dans un tableau VBA
    'ATTENTION: le tableau DOIT ÊTRE DE TYPE Variant
    'Le tableau est automatiquement redimensionné aux dimensions de la plage
    vTableau = Range(.Cells(1, 1), .Cells(1000, 100))

    'Copie d'une matrice VBA dans un tableau Excel
    .Cells.Clear
    .Range(.Cells(1, 1), .Cells(UBound(vTableau, 1), UBound(vTableau, 2))) = vTableau

End With

End Sub
```

Sommaire

Pour traiter un tableau de nombres (ou d'autre type), VBA Excel nous donne le choix d'utiliser une plage Excel ou un tableau VBA. Les avantages de l'un et l'autre sont ici.

Dans les deux cas, on peut utiliser les instructions de boucle pour traiter chaque élément du tableau tour à tour.

Dans une plage excel, il y a plusieurs façons différentes d'adresser chaque cellule.

Dans un tableau VBA, on adresse un élément par sa position (ligne, colonne) dans le tableau.

Fichier exemples

Le fichier Excel Tutoriel Exemples Collections.xlsm contient les exemples présentés dans la présente section.

Suite: Dialogues et formulaires

VBA Excel

par Michel Berthiaume

http://www.usherbrooke.ca/adm/departements/simqg/professeurs/michel-berthiaume/

Dialogues et formulaires

Sur cette page...

Les prochaines pages...

Dialogues, formulaires et alternatives

Les programmes utilisent les dialogues et les formulaires pour obtenir des données ou des instructions de l'utilisateur, pendant l'exécution du programme.

Certains formulaires offrent des menus, permettant à l'utilisateur de choisir entre différentes actions possibles.
En VBA Excel, vous pouvez utiliser une feuille Excel (de préférence la première) à la place de formulaires VBA pour construire des menus.
Cela se fait simplement en insérant des objets (boutons de formulaire par exemple) sur la feuille Excel et en y assignant une macro VBA ou un hyperlien.

Par ailleurs, il se peut que votre programme VBA doive obtenir des instructions ou des données de l'utilisateur.

Si les choix sont simples (continuer ou arrêter, par exemple), il est recommandé d'utiliser la fonction:

MsgBox(*message*[, *boutons*] [, *titre*]**)**

Où:

- *message* est le message affiché par la fonction. On peut utiliser la constante VBA vbCrLf pour insérer des changements de lignes dans un message long.
- *boutons* Expression numérique indiquant les boutons disponibles dans la boîte de dialogue. Par défaut, il n'y a que le bouton Ok.
  Valeurs possibles:
  0 (vbOKOnly) Affiche le bouton OK uniquement.
  1 (vbOKCancel) Affiche les boutons OK et Annuler.
  2 (vbAbortRetryIgnore) Affiche les boutons Abandonner, Réessayer et Ignorer.
  3 (vbYesNoCancel) Affiche les boutons Oui, Non et Annuler.
  4 (vbYesNo) Affiche les boutons Oui et Non.
  5 (vbRetryCancel) Affiche les boutons Réessayer et Annuler.
  16 (vbCritical) Affiche l'icône Message critique.
  32 (vbQuestion) Affiche l'icône Requête d'avertissement.
  48 (vbExclamation) Affiche l'icône Message d'avertissement.
  64 (vbInformation) Affiche l'icône Message d'information.
  256 (vbDefaultButton2) Le deuxième bouton est le bouton par défaut.
  512 (vbDefaultButton3) Le troisième bouton est le bouton par défaut.
  768 (vbDefaultButton4) Le quatrième bouton est le bouton par défaut.
  524288 (vbMsgBoxRight) Le teste est aligné à droite.
  (certaines valeur exotiques sont omises)

  Le premier groupe de valeurs (0 à 5) décrit le nombre et le type de boutons affichés dans la boîte de dialogue. Le deuxième groupe (16, 32, 48 et 64) décrit le style d'icône. Le troisième groupe (256 et 512) définit le bouton par défaut. Au moment d'additionner ces nombres pour obtenir la valeur finale de l'argument buttons, ne sélectionnez qu'un seul nombre dans chaque groupe.
  Les constantes entre parenthèses peuvent être utilisées à la place des valeurs correspondantes et sont plus explicites.
- *titre* titre affiché en haut de la boîte de dialogue.

La fonction **MsgBox()** retourne une valeur différente selon le bouton cliqué pour fermer la boîte de dialogue:

- 1 (vbOK) OK

- 2 (vbCancel) Annuler
- 3 (vbAbort) Abandonner
- 4 (vbRetry) Réessayer
- 5 (vbIgnore) Ignorer
- 6 (vbYes) Oui
- 7 (vbNo) Non

La boîte de dialogue ouverte par l'instruction **MsgBox()** est toujours modale, c'est à dire que l'exécution du programme VBA est interrompu et que le classeur Excel n'est pas accessible tante que la boîte de dialogue ne sera pas fermée par l'utilisateur.

MsgBox() permet au programme d'offrir jusqu'à 3 choix (3 boutons) à l'utilisateur.

Si la boîte de dialogue doit permettre à l'utilisateur de fournir une valeur (et une seule), il est possible d'utiliser l'instruction:

InputBox(*message*[, *titre*] [, *défaut*]**)**

Où:

- *message* est le message affiché par la fonction. On peut utiliser la constante VBA vbCrLf pour insérer des changements de lignes dans un message long.
- *titre* titre affiché en haut de la boîte de dialogue.
- *defaut* valeur qui sera utilisée si l'utilisateur clique sur Ok sans fournir de valeur.

La boîte de dialogue ouverte par l'instruction **InputBox()** est toujours modale, c'est à dire que l'exécution du programme VBA est interrompu et que le classeur Excel n'est pas accessible tante que la boîte de dialogue ne sera pas fermée par l'utilisateur.

InputBox() offre à l'utilisateur une zone de texte et 2 boutons: Ok et Annuler.

Si Excel dispose d'une boîte de dialogue qui convient au besoin, il est possible de les utiliser à partir d'un programme VBA:

Collection **Dialogs**
d'objets **Dialog**

Chaque objet Dialog correspond à une boîte de dialogue Excel.
La méthode **Show** affiche la boîte de dialogue et retourne la réponse fournie par l'utilisateur.
On peut utiliser une constante VBA pour identifier la boîte de dialogue voulue:

Dialogs(xlDialogOpen).Show	Ouvrir fichier
Dialogs(xlDialogSaveAs).Show	Enregistrer sous

Il y a plus de 250 autres boîtes de dialogues disponibles dans Excel. Voir l'aide VBA, ou utiliser l'enregistreuse de macro-commande pour identifier le nom de la boîte de dialogue que vous voulez utiliser.

Il y a aussi l'objet FileDialog:

Objet **FileDialog**

Cet objet permet d'utiliser les boîte de dialogues de fichier standard Ouvrir et Enregistrer de Microsoft Office.

Utilisez la propriété DialogType pour spécifier quelle boîte de dialogue ouvrir:

msoFileDialogFilePicker	Choisir un ou plusieurs fichiers, dont les chemins d'accès sont dans la collection **FileDialogSelectedItems**
msoFileDialogFolderPicker	Choisir un dossier, dont le chemins d'accès est dans la collection **FileDialogSelectedItems**
msoFileDialogOpen	Choisir un ou plusieurs fichiers qui peuvent être ouverts avec la méthode **Execute**
msoFileDialogSaveAs	Choisir un nom de fichier et un chemin d'accès, et d'enregistrer le classeur Excel sous ce nom avec la méthode **Execute**

Plusieurs autres propriétés de cet objet permettent de spécifier le titre, des valeurs par défaut, des filtres, etc.

L'objet FileDialog a 2 méthodes:

- Show qui affiche la boîte de dialogue spécifiée par la propriété DialgType
- Execute qui exécute l'action appropriée pour les dialogues de type **msoFileDialogOpen et msoFileDialogSaveAs**

Pour les besoins plus complexes, comme la saisie de plusieurs valeurs avec une validation importante, vous pouvez utiliser un formulaire VBA.

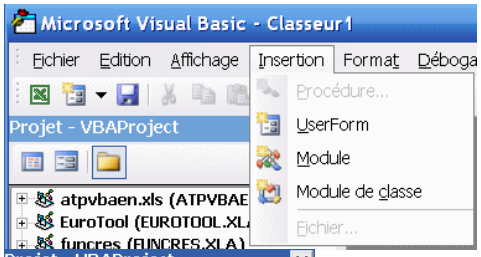
Finalement, pour une saisie vraiment efficace de données vous devriez utiliser Access, dont les formulaires sont beaucoup plus puissants que ceux que vous pouvez programmer en VBA Excel.

L'éditeur de formulaires VBA (MS Forms)

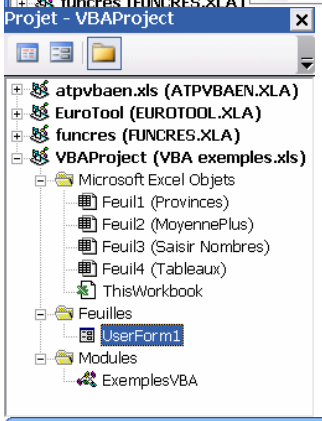
L'éditeur VBA comporte une fenêtre spécifique pour créer ou modifier un formulaire.

Créez un formulaire:

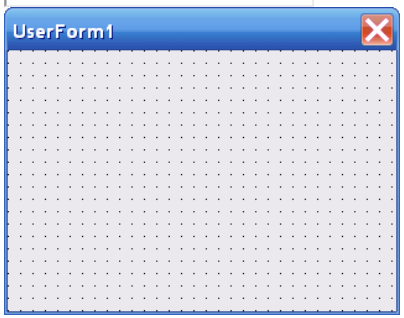
Du menu **Insertion**, sélectionnez **UserForm**



L'explorateur de projet ouvre une branche Feuilles qui ne contient pas des feuilles Excel, mais des formulaires VBA.



La fenêtre principale affiche le cadre d'un formulaire.



C'est l'entête de l'explorateur de projet qui contrôle l'affichage de la fenêtre principale:

 Affichage du code VBA associé au formulaire (une sorte de module incorporé)
Alternatives: Menu Affichage/Code ou F7

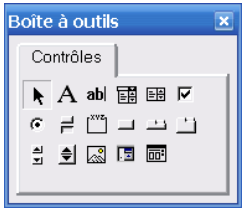
 Affichage du formulaire
Alternatives: Menu Affichage/Objet ou Maj+F7

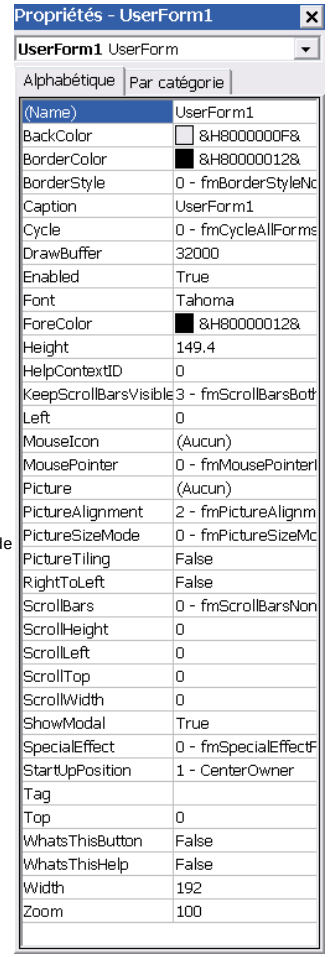
 Modifie le mode d'affichage de l'explorateur de projets. Inutile.

La boîte à outils devrait aussi apparaître.

Sinon, du menu **Affichage**, sélectionnez 

:





Finalement, remarquez sous l'explorateur de projet, la feuille de propriétés du formulaire. C'est là que seront affichées les propriétés de l'objet sélectionné dans le formulaire (ou celles du formulaire lui-même)

Empressez-vous de changer le nom du formulaire que VBA a nommé UserForm1 (ou n).

Il est important de choisir le nom définitif du formulaire AVANT de commencer à lui ajouter du code VBA.

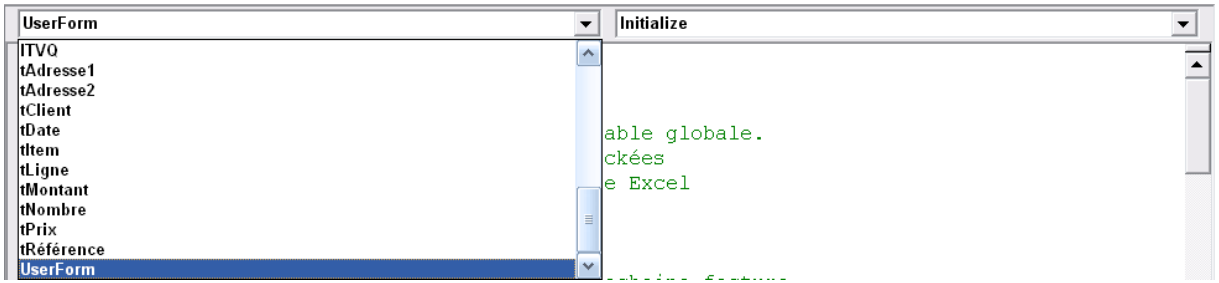
Événements

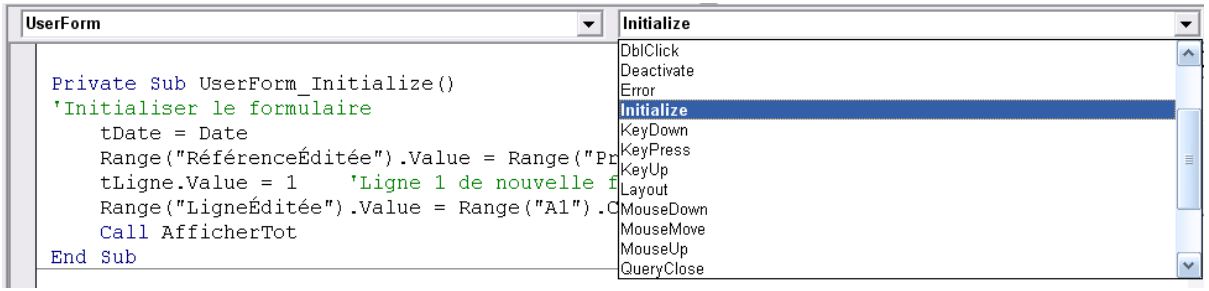
On attache des procédures Sub aux événements que subit le formulaire.

Par exemple, une procédure sub nommée
`Sub UserForm_Initialize()`
est exécutée à chaque fois que le formulaire est ouvert (initialisé).

Pour écrire une procédure événementielle, afficher la fenêtre code du formulaire (

), sélectionner l'objet qui subit l'événement (le formulaire ou un contrôle du formulaire), dans la liste déroulante de gauche, et l'événement voulu dans la liste de droite. L'éditeur VBA crée alors une procédure Sub vide correspondant à l'événement.

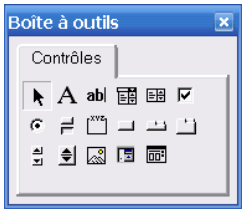




Il est aussi possible de créer une procédure Sub événementielle d'un contrôle en double-cliquant dessus.
C'est alors l'événement par défaut du contrôle qui est choisi.

Contrôles disponibles

On appelle contrôle les classes d'objet qu'on peut insérer dans un formulaire. La description détaillée de chacun de ceux qui sont disponibles en VBA dépasse le cadre du présent texte, mais voici une présentation sommaire de chacun d'entre eux:



	outil de sélection	n'est pas un contrôle, mais l'outil à utiliser pour sélectionner un ou plusieurs contrôles.	
A	libellé (intitulé ou label)	sert à ajouter du texte explicatif dans le formulaire, comme des instructions à l'utilisateur ou des étiquettes pour les zones de texte.	Propriétés intéressantes: • Caption • Value • TextAlign
ab|	zone de texte (textbox)	sert à inclure dans le formulaire une zone dans laquelle l'utilisateur sera invité à entrer des valeurs. La valeur entrée est TOUJOURS enregistrées en format texte, et le programme VBA devra identifier et faire les conversions appropriées.	Propriétés intéressantes: • Name • Enabled • Value • Text • ControlSource • TabIndex • TextAlign Événements intéressants: • Change • Enter • Exit • BeforeUpdate • AfterUpdate
	Zone de liste modifiable (déroulante ou ComboBox)	Sert à inclure dans le formulaire une zone de texte offrant aussi une liste de choix possibles.	Propriétés intéressantes: • Name • Enabled • Visible • Value • TabIndex • Text • TextAlign • BoundColumn • ColumnCount • ControlSource • RowSource

			Événements intéressants: • Change • Enter • Exit • BeforeUpdate • AfterUpdate
	Zone de liste (ListBox)	Sert à inclure dans le formulaire une zone de liste de choix possibles.	Propriétés intéressantes: • Name • Enabled • Visible • Value • TabIndex • Text • TextAlign • BoundColumn • ColumnCount • ControlSource • RowSource Événements intéressants: • Change • Enter • Exit • BeforeUpdate • AfterUpdate
	Case à cocher (CheckBox)	Sert à inclure dans le formulaire une (souvent plusieurs) case(s) à cocher	Propriétés intéressantes: • Name • Caption • Enabled • Visible • Value • TabIndex • TextAlign • ControlSource Événements intéressants: • Change • Enter • Exit • BeforeUpdate • AfterUpdate
	Bouton d'option (OptionButton)	Sert à inclure dans le formulaire un (souvent plusieurs) boutons d'option	Propriétés intéressantes: • Name • Caption • Enabled • Visible • Value • TextAlign • ControlSource Événements intéressants: • Change • Enter • Exit • BeforeUpdate • AfterUpdate
	Bouton Bascule (ToggleButton)	Désuet: utiliser une case à cocher	
	Cadre (Frame)	Sert à regrouper plusieurs contrôles d'un formulaire	Propriétés intéressantes: • Name • Caption • Enabled • Visible Événements intéressants:

			• Enter • Exit
	Bouton de commande (CommandButton)	Sert à inclure dans le formulaire un (souvent plusieurs) bouton(s)	Propriétés intéressantes: • Name • Caption • Default • Enabled • TabIndex • Visible Événements intéressants: • Click
	Contrôle d'onglet (TabStrip)	Sert à inclure des onglets dans le formulaire	Propriétés intéressantes: • Name • Enabled • Value Événements intéressants: • Change
	Multi Page (Page)	Sert à inclure des pages dans le formulaire	Propriétés intéressantes: • Name • Enabled • Value Événements intéressants: • Change
	Barre de défilement (ScrollBar)	Sert à inclure une barre de défilement dans le formulaire.	Propriétés intéressantes: • Name • Enabled • Min • Max • Value • ControlSource • Orientation Événements intéressants: • Change • Enter • Exit • BeforeUpdate • AfterUpdate
	Toupie (SpinButton)	Sert à inclure des flèches de défilement	Propriétés intéressantes: • Name • Enabled • Min • Max • Value • ControlSource • Orientation Événements intéressants: • Change • Enter • Exit • BeforeUpdate • AfterUpdate
	Image	Sert à inclure une image dans le formulaire	Propriétés intéressantes: • Name • Enabled • Picture Événements intéressants: • Click

	RefEdit	Sert à inclure une zone de sélection de plage Excel dans le formulaire Remarque : si il y n'a aucun bouton RefEdit dans la boîte à outils, procédez comme suit : Dans le menu Outils , cliquez sur Autres contrôles . Dans la boîte de dialogue Contrôles supplémentaires , activez la case à cocher RefEdit.ctrl et puis cliquez sur OK.	Propriétés intéressantes: • Name • Enabled • Value • Text Événements intéressants: • Change • Enter • Exit • BeforeDragOver • BeforeUpdate • AfterUpdate

Ouvrir et fermer un formulaire

Pour ouvrir un formulaire, il n'y a pas d'instruction VBA. On utiliser plutôt la méthode **Show** (voir exemples).
Pour réafficher un formulaire déjà ouvert, parce qu'un contrôle a pu changer, ou utilise la méthode **RePaint**.
Pour fermer un formulaire, on peut utiliser la méthode **Close** (ou même **Hide**), mais celle-ci ne libère pas l'espace mémoire occupée par le formulaire.
On utilise plus souvent l'instruction VBA:

Unload *objet*
Où:

- *objet* est le nom du formulaire à supprimer.

Lorsque l'instruction est utilisée dans le module VBA appartenant au formulaire, on préfère utiliser:
Unload Me
L'objet **Me** étant le formulaire en cours.

Exemples

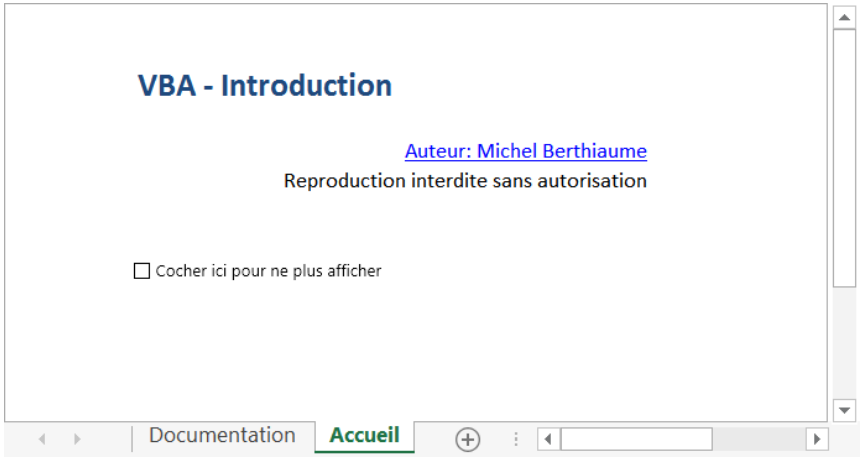
Il est fortement déconseillé d'utiliser quelque forme de dialogue que ce soit dans une fonction (**Function**) VBA.
Une fonction ne devrait communiquer avec l'environnement et l'utilisateur que par ses paramètres (en entrée) et sa valeur (en sortie).
Les exemples ci-dessous ne sont donc que des procédures **Sub**.

Exemple 1: Écran d'accueil

Technique 1: utilisant une feuille Excel

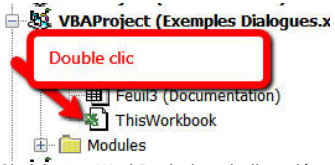
Cet exemple requiert:

- Une feuille Excel nommée Accueil contenant le texte à afficher:

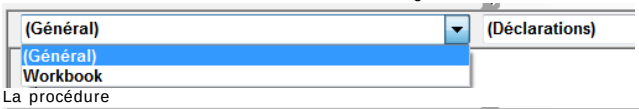


La case à cocher est liée à une cellule nommée Accueil, qui est elle-même en blanc sur fond blanc pour que le contenu ne soit pas visible à l'écran.

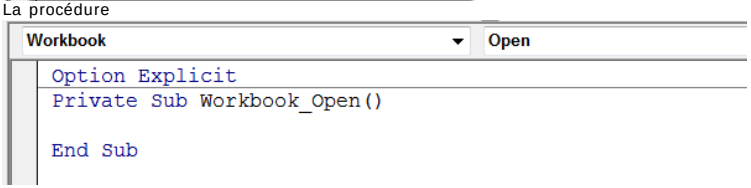
- La procédure événementielle Sub Workbook_Open(), créée en
  - Double-cliquant sur ThisWorbook (fenêtre de projet de l'éditeur VBA)



- Choisissant WorkBook dans la liste déroulante de gauche de la fenêtre de code



- La procédure



apparaît alors..

- Complétez la ainsi:

```
Option Explicit
Private Sub Workbook_Open()
'Auteur Michel Berthiaume
'Afficher une feuille Excel de présentation de l'application

Dim dtDate As Date

On Error GoTo Erreur:

    If Sheets("Accueil").Range("Accueil").Value = False Then 'Si l'utilisateur n'a rien  coché
        Sheets("Accueil").Visible = True
        Sheets("Accueil").Select
        dtDate = Now()
        Do Until Now() - dtDate > (1 / 24 / 60 / 60 * 3) 'Boucle d'attente de 3 secondes
            DoEvents  'Permet à l'utilisateur de cocher la case
        Loop
    End If
    Sheets("Accueil").Visible = False

    Exit Sub
Erreur:
    MsgBox "Erreur inattendue: " & Err.Description
End Sub
```

La feuille Accueil sera affichée lors de l'ouverture du classeur.

Technique 2: utilisant un formulaire

Note: le document VBA Excel - Créer un formulaire (User Form) d'accueil décrit les étapes de création du formulaire ci-dessous.

Cet exemple requiert:

- Un formulaire VBA Excel (Microsoft Forms) nommé frmAccueil contenant le texte à afficher:

  dont la propriété

  - Name: frmAccueil
  - Caption: Accueil

  Contenant les contrôles (objets) suivants:

- Une image (ici une capture de la feuille d'accueil Excel de l'exemple 1)
- Une case à cocher (CheckBox):
  -

Name: cbMessage
- Caption: Ne plus afficher ce message
- ControlSource: =Accueil

- La procédure événementielle Sub Workbook_Open():
  - Double-cliquez sur ThisWorbook (fenêtre de projet de l'éditeur VBA)
  - Choisissez WorkBook dans la liste déroulante de gauche de la fenêtre de code
  - La procédure

```
Option Explicit
Private Sub Workbook_Open()

End Sub
```

apparaît alors.

  - Complétez la ainsi:

```
Option Explicit
Private Sub Workbook_Open()
'Auteur Michel Berthiaume
'Afficher un formulaire VBA Excel (Microsoft Forms) de présentation de l'application
  frmAccueil.show

End Sub
```

- La procédure événementielle UserForm_Activate():
  - Affichez le fomulaire frmAccueil
  - Basculez en mode Code (bouton  dans la fenêtre Projet, Menu Affichage/Code ou )
  - Choisissez UserForm dans la liste déroulante de gauche de la fenêtre de code
  - Choisissez Activate dans la liste déroulante de droite de la fenêtre de code
  - La procédure

```
Option Explicit
Private Sub UserForm_Activate()

End Sub
```

apparaît alors.

  - Complétez la ainsi:

```
Option Explicit
Private Sub UserForm_Activate()
'Auteur: Michel Berthiaume
'Accueil dans un classeur Excel
Dim dtDate As Date

On Error GoTo Erreur:
    If Me.cbMessage.Value = False Then 'Seulement si la case n'est pas cochée
        Me.Repaint 'Afficher le formulaire
        dtDate = Now()
        Do Until (Now() - dtDate) > (1 / 24 / 60 / 60 * 3) 'Boucle d'attente de 3 secondes
            DoEvents 'Permet à l'utilisateur de cocher la case
        Loop
    End If
    Unload Me 'Effacer le formulaire
    Exit Sub

Erreur:
    MsgBox "Erreur inattendue: " & Err.Description
End Sub
```

Exemple 2: Sélectionner un dossier

L'exemple ci-dessous crée une liste de fichiers d'un dossier choisi par l'utilisateur.

Il requiert une feuille Excel contenant des plages nommées:

- NomDossier, qui contiendra le chemin du dossier choisi
- ListeFichiers qui est la cellule de gauche du titre de la plage qui contiendra la liste.

```
Sub ListeFichiers()
'Auteur: Michel Berthiaume
'Créer une feuille Excel contenant la liste des fichiers d'un dossier choisi par l'utilisateur
Dim FSO As New Scripting.FileSystemObject 'Requiert une référence à Microsoft Scripting Runtime
Dim fsFichier As File
Dim lLigne As Long

On Error GoTo Erreur:

    'Obtenir le nom du dossier
    With Application.FileDialog(msoFileDialogFolderPicker)
        .show 'Afficher la boîte de dialogue
        If .SelectedItems.Count <> 1 Then 'Opération annulée
            Exit Sub
        End If
        Range("NomDossier").Value = .SelectedItems(1) 'Inscrire dans la feuille
```

```
    End With

    'Inscrire les fichiers
    With Range("ListeFichiers")
        'Effacer les données présentes
         .Range(.Offset(0, 0), .Offset(.CurrentRegion.Rows.Count, _
             .CurrentRegion.Columns.Count)).Clear

        lLigne = 1 'Offset initial
        For Each fsFichier In FSO.GetFolder(Range("NomDossier").Value).Files
            .Offset(lLigne, 0).Value = fsFichier.Name
            .Offset(lLigne, 1).Value = fsFichier.Size
            .Offset(lLigne, 2).Value = fsFichier.DateLastModified
            lLigne = lLigne + 1
        Next
    End With

    Exit Sub
Erreur:
    MsgBox "Erreur inattendue: " & Err.Description
End Sub
```

Commentaires:

- Notez l'utilisation de l'objet FileSystemObject de la librairie Scripting pour parcourir le dossier choisi.

Exemple 3: Barre de progression

L'exemple précédent peut être long à exécuter, si le dossier choisi contient beaucoup de fichiers.
Ajoutons une barre de progression.

Technique 1: En utilisant la ligne d'état

Excel permet d'afficher un message dans la ligne d'état en bas le l'écran, en utilisant la propriété StatusBar.

Un programme peut appeler la procédure suivante pour afficher un % de progression. Il suffit qu'il rappelle la procédure à chaque fois que le % est modifié.

```
Sub AfficheProgression(PourcentageEffectué)
'Affiche un pourcentage et une barre de progression dans la barre d'état Excel
'PourcentageEffectué doit être entre 0 et 1 (inclus)
'Si 0, négatif ou >1, efface barre d'état
'Par Michel Berthiaume
'Inspiré du programme de Dick Kusleika

    If IsNumeric(PourcentageEffectué) And _
       (PourcentageEffectué > 0) And _
       (PourcentageEffectué <= 1) Then
        Application.StatusBar = Format(PourcentageEffectué, "0%") & " " & String(PourcentageEffectué * 50, Chr(31))
    Else
        Application.StatusBar = False
    End If
End Sub
```

Commentaires:

- Cette procédure requiert un paramètre qui a 2 utilisations:
  - entre 0 et 1, il indique le % de progression à afficher
  - négatif, nul ou >1, il efface le contenu de StatusBar.
- La fonction VBA String() construit la barre de progression
- La fonction Chr(31) affiche le caractère , qui sert à construire la barre de progression.

On modifie donc le programme de l'exemple précédent ainsi:

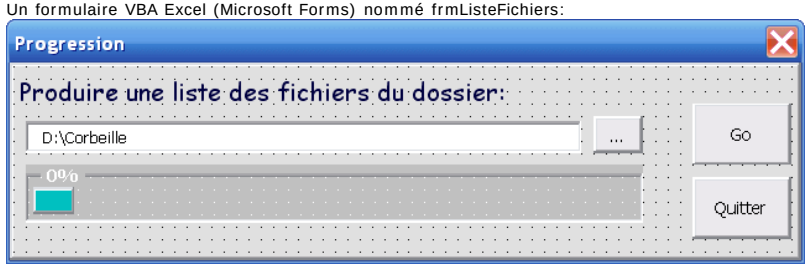
```
Sub ListeFichiers1()
'Auteur: Michel Berthiaume
'Créer une feuille Excel contenant la liste des fichiers d'un dossier choisi par l'utilisateur
DimFSO As New Scripting.FileSystemObject 'Requiert une référence à Microsoft Scripting Runtime
Dim fsFichier As File
Dim lLigne As Long
Dim lNombre As Long 'Nombre de fichiers


    'Obtenir le nom du dossier
    With Application.FileDialog(msoFileDialogFolderPicker)
        .show 'Afficher la boîte de dialogue
        If .SelectedItems.Count <> 1 Then 'Opération annulée
            Range("NomDossier").Value = ""
            Exit Sub
         End If
        Range("NomDossier").Value = .SelectedItems(1) 'Inscrire dans la feuille
    End With

    lNombre = FSO.GetFolder(Range("NomDossier").Value).Files.Count

    'Inscrire les fichiers
    With Range("ListeFichiers")
        'Effacer les données présentes
         .Range(.Offset(0, 0), .Offset(.CurrentRegion.Rows.Count, _
             .CurrentRegion.Columns.Count)).Clear

        lLigne = 1 'Offset initial
        For Each fsFichier In FSO.GetFolder(Range("NomDossier").Value).Files
            Call AfficheProgression(lLigne / lNombre) 'Afficher progression
            .Offset(lLigne, 0).Value = fsFichier.Name
            .Offset(lLigne, 1).Value = fsFichier.Size
            .Offset(lLigne, 2).Value = fsFichier.DateLastModified
            lLigne = lLigne + 1
        Next
    End With
```

```
    Call AfficheProgression(0) 'Effacer la barre de progression
End Sub

Sub AfficheProgression(PourcentageEffectué)
'Affiche un pourcentage et une barre de progression dans la barre d'état Excel
'PourcentageEffectué doit être entre 0 et 1 (inclus)
'Si 0, négatif ou >1, efface barre d'état
'Par Michel Berthiaume
'Inspiré du programme de Dick Kusleika

    If PourcentageEffectué > 0 And PourcentageEffectué <= 1 Then
        Application.StatusBar = Format(PourcentageEffectué, "0%") & " " & String(PourcentageEffectué * 50, Chr(31))
    Else
        Application.StatusBar = False
    End If

End Sub
```

Technique 2: En utilisant un formulaire

Cet technique requiert:

- Un formulaire VBA Excel (Microsoft Forms) nommé frmListeFichiers:



  dont la propriété
  - Caption : Progression

  Contenant les contrôles (objets) suivants:
- Une étiquette (Label):
  - Caption: Produire une liste des fichiers du dossier:
- Une zone de texte (TextBox):
  - Name: tDossier
- Un cadre (Frame):
  - Name: FrameProgress
  - Caption: 0%

    contenant:
    - Une étiquette (Label):
      - Name: LabelProgress
      - BackColor: &H00C0C000&
- Trois boutons (CommandButton):
  - Name: bChoisirDossier, Caption: ...
  - Name: bGo, Caption: Go
  - Name: bQuitter, Caption: Quitter

Il faut associer au formulaire (et à ses contrôles) le code VBA suivant:

```
Option Explicit
'Auteur: Michel Berthiaume
'Inspiré du travail de John Walkenbach
'Formulaire de production de liste de fichiers

Private Sub UserForm_Initialize()
'Initialiser le formulaire
    If tDossier.Value = "" Then
        tDossier.Value = ActiveWorkbook.Path 'Dossier par défaut
    End If
End Sub

Private Sub bChoisirDossier_Click()
'Obtenir le nom du dossier
    With Application.FileDialog(msoFileDialogFolderPicker)
        .Show 'Afficher la boîte de dialogue
        If .SelectedItems.Count <> 1 Then 'Opération annulée
            Range("NomDossier").Value = ""
            Exit Sub
        End If
        tDossier.Value = .SelectedItems(1) 'Inscrire dans le formulaire et la feuille
    End With
End Sub

Private Sub bGo_Click()
'Produire la liste des fichiers
'Inscrire les fichiers
Dim FSO As New Scripting.FileSystemObject 'Requiert une référence à Microsoft Scripting Runtime
Dim fsFichier As File
Dim lLigne As Long
Dim lNombre As Long 'Nombre de fichiers
```

```
    On Error GoTo DossierInexistant
    lNombre = FSO.GetFolder(Range("NomDossier").Value).Files.Count
    On Error GoTo 0
    If lNombre = 0 Then
        Exit Sub
    End If

With Range("ListeFichiers")
    'Effacer les données présentes
    .Range(Cells(2, 1), Cells(.CurrentRegion.Rows.Count - 1, _
        .CurrentRegion.Columns.Count)).Clear

    lLigne = 1 'Offset initial
    For Each fsFichier In FSO.GetFolder(Range("NomDossier").Value).Files
        Call AfficheProgression(lLigne / lNombre) 'Afficher progression
        .Offset(lLigne, 0).Value = fsFichier.Name
        .Offset(lLigne, 1).Value = fsFichier.Size
        .Offset(lLigne, 2).Value = fsFichier.DateLastModified
        lLigne = lLigne + 1
    Next
End With
Exit Sub

DossierInexistant:
    MsgBox "Dossier inexistant", , "Liste de fichiers d'un dossier"
End Sub

Private Sub bQuitter_Click()
    Unload Me
End Sub

Sub AfficheProgression(PourcentageEffectue)
    FrameProgress.Caption = Format(PourcentageEffectue, "0%")
    LabelProgress.Width = PourcentageEffectue * (FrameProgress.Width - 10)
    Me.Repaint
End Sub
```

Finalement, ajouter la procédure:

```
Sub OuvrirfrmListeFichiers()
    FrmListeFichiers.Show
End Sub
```

dans un module.

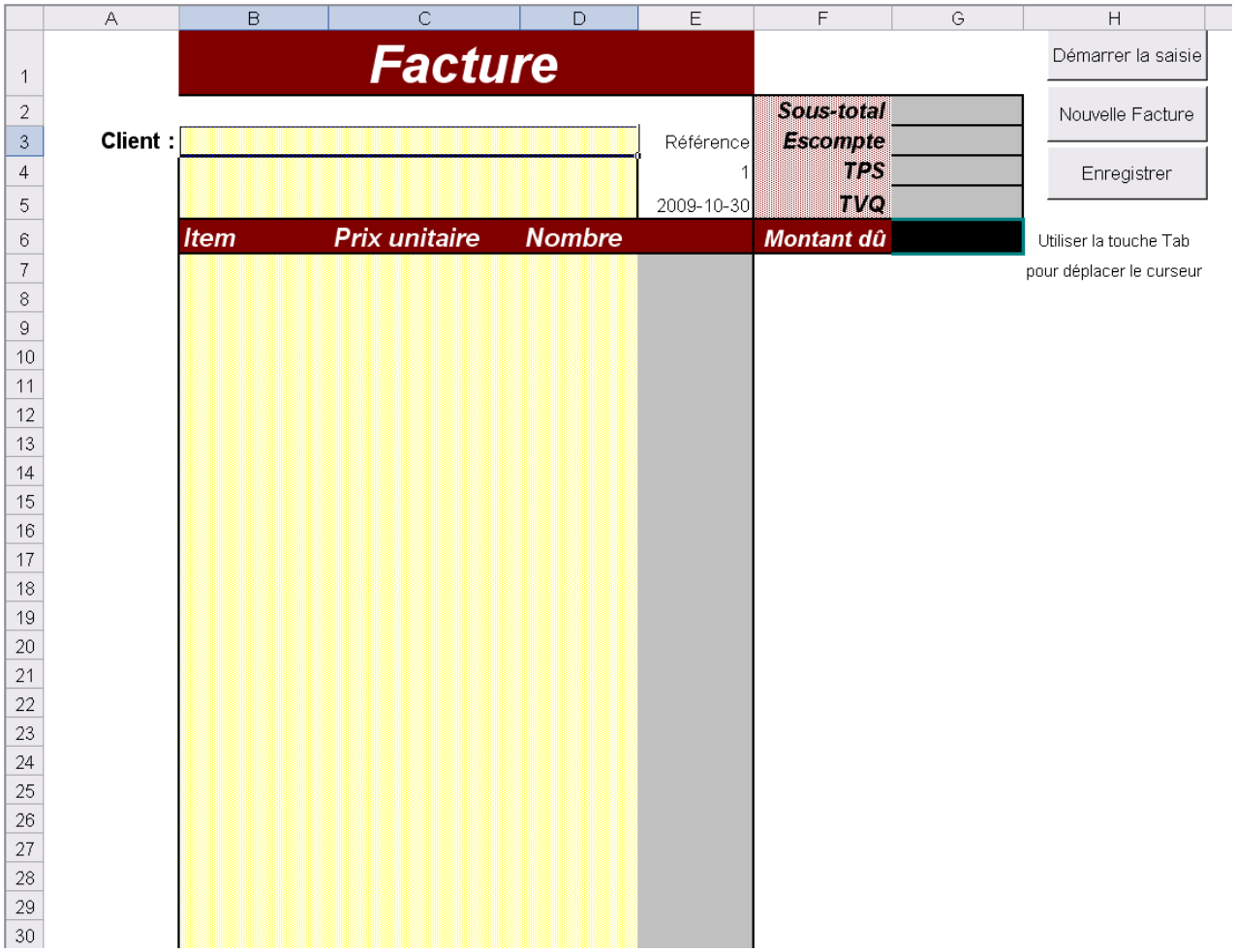
Lorsque exécutée, cette dernière procédure exécutera le formulaire.

Exemple 4: Saisir et valider des données

Voyons comment on peut utiliser Excel pour saisir les informations nécessaires à une facturation de vente.

Technique 1: utilisant une feuille Excel

Créez la feuille Excel suivante:



Ayant les caractéristiques suivantes:

- B3 à D5 déverrouillées (saisie du client)
- B7 à D48 déverrouillées (saisie des ventes)
- G2 à G6 les calculs de montant de facture
- Plage nommée NomClient: B3
- Plage nommée Référence: E4
- Protection de la feuille activée

Chacun des trois boutons est associé au procédures correspondantes:

```
Sub Initial()
'Auteur: Michel Berthiaume
'Initialiser ou réinitialiser la feuille facture
'l'action est déterminée par le contenu du bouton: Démarrer la saisie ou Terminer la saisie
'Configuré pour Excel 2007
Dim sBouton As Shape

    'Trouver le bouton
    For Each sBouton In ActiveSheet.Shapes 'Examiner chaque bouton

        If sBouton.AlternativeText = "Démarrer la saisie" Then

            'Initialiser
            With ActiveWindow 'Déactive certains affichages
                .DisplayHeadings = False
                .DisplayHorizontalScrollBar = False
                .DisplayVerticalScrollBar = False
                .DisplayWorkbookTabs = False
            End With
            With Application
                .DisplayFormulaBar = False
                .DisplayStatusBar = False
            End With
```

```
            'Modifier le bouton
            Worksheets("Facture").Unprotect
            sBouton.AlternativeText = "Terminer la saisie"
            sBouton.Select                          'Impossible de modifier l'étiquette directement
            Selection.Characters.Text = "Terminer la saisie"
            Range("NomClient").Select
            Worksheets("Facture").Protect
            Exit For 'Pas besoin de regarder les autres boutons

        ElseIf sBouton.AlternativeText = "Terminer la saisie" Then

            'Rétablir l'affichage
            With ActiveWindow
                .DisplayHeadings = True
                .DisplayHorizontalScrollBar = True
                .DisplayVerticalScrollBar = True
                .DisplayWorkbookTabs = True
            End With
            With Application
                .DisplayFormulaBar = True
                .DisplayStatusBar = True
            End With
            'Modifier le bouton
            Worksheets("Facture").Unprotect
            sBouton.AlternativeText = "Démarrer la saisie"
            sBouton.Select                          'Impossible de modifier l'étiquette directement
            Selection.Characters.Text = "Démarrer la saisie"
            Range("NomClient").Select
            Worksheets("Facture").Protect
            Exit For 'Pas besoin de regarder les autres boutons
        End If
    Next
End Sub

Sub Nouvelle()
'Auteur: Michel Berthiaume
'Prépare la feuille pour une nouvelle facture

    Range("B3:D5").ClearContents
    Range("B7:D16").ClearContents
    Worksheets("Facture").Unprotect
    Range("Référence") = Range("Référence") + 1
    Worksheets("Facture").Protect
    Range("NomClient").Select
End Sub

Sub Enregistrer()
'Auteur: Michel Berthiaume
'Exécuté par bouton Ok de boîte de dialogue

    With Application.FileDialog(msoFileDialogSaveAs)
        .InitialFileName = "Facture de " & Range("NomClient").Value & " du " & Format(Date, "Long Date") & ".xls"
        .Show 'Afficher la boîte de dialogue
        If .SelectedItems.Count <> 1 Then 'Opération annulée
            Exit Sub
        End If
        ActiveWorkbook.SaveAs Filename:=.SelectedItems(1)
    End With

End Sub
```

Technique 2: utilisant un formulaire

Ici, les factures seront enregistrées dans une liste Excel.
Chaque facture est identifiée par un numéro de référence (séquentiel).
Chaque facture est décrite sur plusieurs lignes:

- la ligne 0 identifie le client
- Les lignes 1 à 89 correspondent aux lignes de détail de la facture (items).
- Les lignes 91 à 99 contiennent le sous-total, les taxes et le total de la facture.

Cette liste de factures est facilement utilisable avec les filtres, le tris et le tableau croisé dynamique d'Excel.

Créez la feuille Excel selon le modèle ci dessous:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	Référence	Date	Ligne	Client	Adresse 1	Adresse 2	Item	Prix unitaire	Nombre	Montant		Prochaine référence:	1			
2											Saisir facture	Ligne éditée:	2	Numéro de ligne de la liste		
3												Référence éditée:	1	Numéro de facture en cours		
4												Sous-total:	- $			
5												TPS:	- $	5.00%		
6												TVQ:	- $	7.50%		
7												Total:	- $			
8																

La partie gauche contiendra la liste de facture.

Au milieu (colonne K), le bouton qui appelle le formulaire.

À droite, les colonnes L, M et N contiennent des valeurs qui seront utilisées par le formulaire.

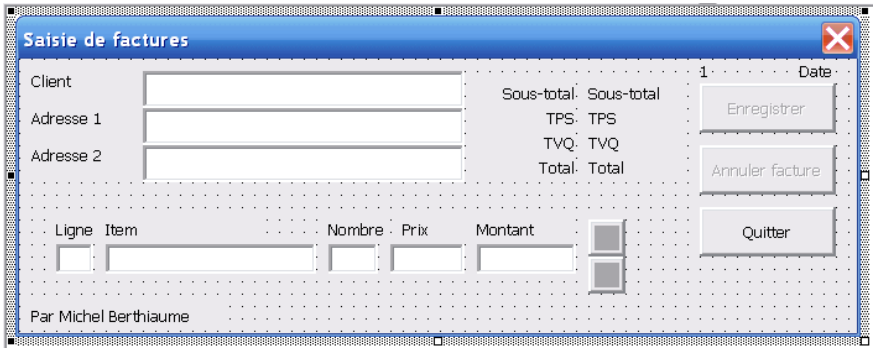
Pour que le bouton Saisir factures ouvre le formulaire:

- Créez une procédure Sub:

```
Sub OuvrirfrmFacture()
    frmFacture.Show
End Sub
```

- Associez cette procédure au bouton (clic droit sur le bouton, affecter une macro...)

Le formulaire frmFacture est conçu de la façon suivante:



Le formulaire compte les zones de texte suivantes:

Nom	Propriétés importantes	Description
tClient	TabIndex: 0	Contiendra le nom du client
tAdresse1	TabIndex: 1	Ligne 1 de l'adresse du client
tAdresse2	TabIndex: 2	Ligne 2 de l'adresse du client
tLigne	Enabled: False	Numéro de la ligne dans la facture
tItem	TabIndex: 4	Description de l'item acheté
tNombre	TabIndex: 5	Nombre d'items achetés
tPrix	TabIndex: 6	Prix unitaire
tMontant	Enabled: False	Extension (Nombre * Prix)

Notez que les zones tLignes et tMontant, qui sont gérées par le programme, sont désactivées (Enabled: False).

Chacune de ces zones de texte est accompagnée d'une étiquette (Lablel) qui la décrit.

Il y a aussi les 8 étiquettes qui décrivent les montants de la facture. 4 d'entre elles sont descriptives et ont les noms assignés lors de leur création (Label15, Label16, etc.).
Les 4 autres seront utilisées par le programme pour afficher les valeurs et se nomment: lSousTotal, lTPS, lTVQ et lTotal.

Deux autres étiquettes, tRéférence et tDate afficheront le numéro de la facture affichée et sa date.

Finalement, le formulaire compte 5 boutons:

Nom	Propriétés importantes	Description
bLignePréc	TabIndex: 8 Enabled: False Picture: Image d'un triangle	Enregistre la ligne affichée. Permet de reculer d'une ligne dans la facture.
bLigneSuiv	TabIndex: 9 Enabled: False Picture: Image d'un triangle	Enregistre la ligne affichée. Permet de passer à la ligne suivante dans la facture
bEnregistrer	TabIndex: 10 Caption: Enregistrer Enabled: False	Enregistre la facture.
bAnnuler	TabIndex: 11 Caption: Annuler facture Enabled: False	Annule la facture (et les lignes enregistrées)
bQuitter	TabIndex: 12 Caption: Quitter Enabled: True	Ferme le formulaire

Le code VBA associé à ce formulaire peut être groupé en quelques blocs:

D'abord la documentation:

```
Option Explicit
'Auteur: Michel Berthiaume
'Formulaire d'enregistrement de factures
'Noter que le programme n'utilise aucune variable globale.
'Les données externes aux procédures sont stockées
'Soit dans le formulaire, soit dans la feuille Excel
'Ce qui les rend plus accessibles.
'
'Données critiques:
'Range("ProchaineRéférence"): numéro de la prochaine facture
'Range("RéférenceEditée"): numéro de la facture en cours (synchrone avec la suivante)
'frmFacture.tRéférence: numéro de la facture en cours (synchrone avec la précédente)
'Range("LigneEditée"): numéro de la ligne en cours de la feuille Excel
'frmFacture.tLigne: numéro de ligne en cours de la facture en cours
```

- Ce programme ne contient aucune variable globale. Toutes les données partagées entre les différentes procédures sont stockées soit dans la feuille Excel, soit dans le formulaire. Cette façon de faire expose les données à

l'utilisateur et au programmeur, ce qui rend plus aisé le suivi du déroulement du programme.
Il faut aussi noter que c'est la fonction Excel Somme.Si() qui est utilisée (dans la feuille Excel) pour calculer le sous-total de la facture en cours. Ceci permet d'éviter du code VBA et facilite encore une fois une vérification visuelle des résultats.

L'initialisation:

```
Private Sub UserForm_Initialize()
'Initialiser le formulaire
    tDate = Date
    Range("RéférenceÉditée").Value = Range("ProchaineRéférence").Value
    tLigne.Value = 1 'Ligne 1 de nouvelle facture
    Range("LigneÉditée").Value = Range("A1").CurrentRegion.Rows.Count + 1 'Première ligne vide
    Call AfficherTot
End Sub
```

Commentaires:

- Cette procédure est exécutée au chargement (load) du formulaire.
- Les champs Date, Référence et numéro de ligne sont initialisées
- Le pointeur de ligne dans Excel est intitialisé
- La procédure qui affiche les totaux de facture est exécutée.

Le traitement des lignes de facture:

```
Private Sub tNombre_Change()
    'Recalculer montant
    tMontant.Value = Val(tNombre.Value) * Val(tPrix.Value)
End Sub

Private Sub tPrix_Change()
    'Recalculer montant
    tMontant.Value = Val(tNombre.Value) * Val(tPrix.Value)
End Sub

Private Sub tMontant_Change()
    'Activer les boutons de changement de ligne

    If tMontant.Value <> 0 Then
        bLigneSuiv.Enabled = True
    Else
        bLigneSuiv.Enabled = False
    End If
    If tLigne.Value = 1 Then
        bLignePréc.Enabled = False
    Else
        bLignePréc.Enabled = True
    End If
End Sub

Private Sub bLignePréc_Click()
    If tLigne.Value = 1 Then
        Beep 'On ne peut pas reculer avant la ligne 1
        Exit Sub
    End If

    'Traiter la ligne en cours
    If tMontant.Value <> 0 Then 'Enregistrer la ligne s'il y a un montant
        Cells(Range("LigneÉditée").Value, 1) = tRéférence.Value
        Cells(Range("LigneÉditée").Value, 2) = tDate.Value
        Cells(Range("LigneÉditée").Value, 3) = tLigne.Value
        Cells(Range("LigneÉditée").Value, 7) = tItem.Value
        Cells(Range("LigneÉditée").Value, 8) = tPrix.Value
        Cells(Range("LigneÉditée").Value, 9) = tNombre.Value
        Cells(Range("LigneÉditée").Value, 10) = tMontant.Value
    End If

    'Passer à la ligne précédente
    Range("LigneÉditée").Value = Range("LigneÉditée").Value - 1

    'Édition de ligne existante
    tRéférence.Value = Cells(Range("LigneÉditée").Value, 1)
    tDate.Value = Cells(Range("LigneÉditée").Value, 2)
    tLigne.Value = Cells(Range("LigneÉditée").Value, 3)
    tItem.Value = Cells(Range("LigneÉditée").Value, 7)
    tPrix.Value = Cells(Range("LigneÉditée").Value, 8)
    tNombre.Value = Cells(Range("LigneÉditée").Value, 9)
    Call AfficherTot
    tItem.SetFocus 'Activer le 1er champ
End Sub

Private Sub bLigneSuiv_Click()
    'Traiter la ligne en cours
    If Val(tMontant.Value) = 0 Then 'Ne pas enregistrer ligne nulle
        Beep
        Exit Sub
    Else 'enregistrer la ligne
        Cells(Range("LigneÉditée").Value, 1) = tRéférence.Value
        Cells(Range("LigneÉditée").Value, 2) = tDate.Value
        Cells(Range("LigneÉditée").Value, 3) = tLigne.Value
        Cells(Range("LigneÉditée").Value, 7) = tItem.Value
        Cells(Range("LigneÉditée").Value, 8) = tPrix.Value
        Cells(Range("LigneÉditée").Value, 9) = tNombre.Value
        Cells(Range("LigneÉditée").Value, 10) = tMontant.Value
    End If

    'Passer à la ligne suivante
    Range("LigneÉditée").Value = Range("LigneÉditée").Value + 1

    If Range("LigneÉditée").Value > 899 Then 'Pas plus de 899 lignes par facture
        Range("LigneÉditée").Value = 899
    End If
```

```
    'Initialiser les champs
    If Cells(Range("LigneÉditée").Value, 1) <> "" Then
        'Édition de ligne existante
        tRéférence.Value = Cells(Range("LigneÉditée").Value, 1)
        tDate.Value = Cells(Range("LigneÉditée").Value, 2)
        tLigne.Value = Cells(Range("LigneÉditée").Value, 3)
        tItem.Value = Cells(Range("LigneÉditée").Value, 7)
        tPrix.Value = Cells(Range("LigneÉditée").Value, 8)
        tNombre.Value = Cells(Range("LigneÉditée").Value, 9)
        Call AfficherTot
    Else
        Call NouvelleLigne
    End If
    tItem.SetFocus 'Activer le 1er champ
End Sub
```

Commentaires:

- Les 2 premières procédures sont déclenchées par des changement au contenu des textbox tNombre ou tPrix et assurent le recalcul des montants lorsque le nombre ou le prix de l'item sont modifiés.
- La suivante (tMontant_Change()) est déclenchée par une ou l'autre des deux précédentes, et active ou désactive les boutons appropriés. Les deux dernières sont déclenchées par les boutons de déplacement entre les lignes de la facture et assurent:
  - l'enregistrement de la ligne affichée
  - le passage à la ligne précédente ou suivante, selon le cas.
- Notez l'utilisation de SetFocus pour modifier l'ordre de tabulation des contrôles.

Le traitement de facture:

```
Private Sub bEnregistrer_Click()
'Enregistrer la facture
Dim lLigne As Long

    If Val(tMontant.Value) <> 0 Then 'Enregistrer ligne en cours au besoin
        bLigneSuiv_Click
    End If

    lLigne = Range("LigneÉditée").Value = Range("LigneÉditée").Value

    'Information client
    Cells(lLigne, 1).Value = tRéférence.Value
    Cells(lLigne, 2).Value = tDate.Value
    Cells(lLigne, 3).Value = 0
    Cells(lLigne, 4).Value = tClient.Value
    Cells(lLigne, 5).Value = tAdresse1.Value
    Cells(lLigne, 6).Value = tAdresse2.Value

    'Sous-total
    lLigne = lLigne + 1
    Cells(lLigne, 1).Value = tRéférence.Value
    Cells(lLigne, 2).Value = tDate.Value
    Cells(lLigne, 3).Value = 900
    Cells(lLigne, 7).Value = "Sous-total"
    Cells(lLigne, 10).Value = Range("SousTotal").Value

    'TPS
    lLigne = lLigne + 1
    Cells(lLigne, 1).Value = tRéférence.Value
    Cells(lLigne, 2).Value = tDate.Value
    Cells(lLigne, 3).Value = 901
    Cells(lLigne, 7).Value = "TPS"
    Cells(lLigne, 10).Value = Round(Range("TPS").Value, 2)

    'TVQ
    lLigne = lLigne + 1
    Cells(lLigne, 1).Value = tRéférence.Value
    Cells(lLigne, 2).Value = tDate.Value
    Cells(lLigne, 3).Value = 902
    Cells(lLigne, 7).Value = "TVQ"
    Cells(lLigne, 10).Value = Round(Range("TVQ").Value, 2)

    'Total
    lLigne = lLigne + 1
    Cells(lLigne, 1).Value = tRéférence.Value
    Cells(lLigne, 2).Value = tDate.Value
    Cells(lLigne, 3).Value = 999
    Cells(lLigne, 7).Value = "Total"
    Cells(lLigne, 10).Value = Round(Range("Total").Value, 2)

    'Prochaine facture
    Range("RéférenceÉditée").Value = Range("ProchaineRéférence").Value

    'Prochaine ligne
    tLigne.Value = 0
    Call NouvelleLigne 'Sera ligne 1

    bAnnuler.Enabled = False
    bEnregistrer.Enabled = False

End Sub

Private Sub bAnnuler_Click()
    If MsgBox("Abandonner la facture en cours?", vbOKCancel, "Annuler facture") = vbOK Then
        Call annuler
    End If
End Sub

Private Sub bQuitter_Click()
    If Range("Total").Value <> 0 Then
        If MsgBox("Abandonner la facture en cours?", vbOKCancel, "Quitter sans enregistrer?") = vbOK Then
            Call annuler
            Unload Me
```

```
        End If
    Else
        Unload Me
    End If
End Sub
```

Commentaires:

- Ces procédures correspondent aux trois boutons de traitement de facture, Enregistrer, Annuler et Quitter.
- Ces boutons ne sont actifs que si le contexte le permet (géré par la procédure tMontant_Change()
- Le bouton Enregistrer déclenche:
  - l'enregistrement de la ligne en cours (en utilisant la procédure bLigneSuiv_Click)
  - l'ajout de 5 lignes à la liste de données pour stocker l'information de la facture
  - l'initialisation de différents contrôles et boutons.
- Le bouton Annuler efface les lignes qui ont été enregistrées pour la facture en cours
- Le bouton Quitter
  - s'assure d'enregistrer (ou non) la facture en cours
  - ferme le formulaire.

Les procédures utilitaires:

```
Sub annuler()
Dim lLigne As Long

    If tLigne.Value = 1 Then 'Pas de facture
        'Annuler ligne en cours
        tLigne.Value = 0
        Call NouvelleLigne
        Exit Sub
    End If

    'Annuler les lignes de la facture annulée
    For lLigne = Val(Range("LigneÉditée").Value) - 1 To 1 Step -1
        If Cells(lLigne, 1).Value <> Val(tRéférence.Value) Then
            Exit For 'Terminé
        Else
            Range(Cells(lLigne, 1), Cells(lLigne, 10)).ClearContents
        End If
    Next

    'Annuler ligne en cours
    tLigne.Value = 0
    Call NouvelleLigne

    'Annuler entête
    tClient.Value = ""
    tAdresse1.Value = ""
    tAdresse2.Value = ""
End Sub

Sub NouvelleLigne()
    tLigne.Value = tLigne.Value + 1
    tItem.Value = ""
    tPrix.Value = ""
    tNombre.Value = ""
    Call AfficherTot
End Sub

Sub AfficherTot()
'Afficher les totaux de facture
'et activer les boutons appropriés.

    Calculate 'S'assurer d'avoir les bonnes valeurs
    lSousTotal.Caption = Range("SousTotal").Text
    lTPS.Caption = Range("TPS").Text
    lTVQ.Caption = Range("TVQ").Text
    lTotal.Caption = Range("Total").Text
    If Range("Total").Value > 0 Then
        bAnnuler.Enabled = True
        bEnregistrer.Enabled = True
    Else
        bAnnuler.Enabled = False
        bEnregistrer.Enabled = False
    End If
End Sub
```

Commentaires: Ces procédures contiennent des opérations qui sont utilisées par plusieurs autres procédures du formulaire.
On préfère regrouper ces opérations plutôt que les copier dans les différentes procédures qui les utilisent.

- Annuler() est utilisée par bAnnuler_Click() et bQuitter_Click()
- NouvelleLigne() est utilisée par Annuler() et bLigneSuiv_Click()
- AfficherTot() est utilisée par UserForm_Initialize(), bLignePréc_Click(), bLigneSuiv_Click() et NouvelleLigne().

Sommaire

Fichier Exemples

Suite: Objets et événements Excel

VBA Excel

par Michel Berthiaume

http://www.usherbrooke.ca/adm/departements/simqg/professeurs/michel-berthiaume/

Objets et événements Excel

Sur cette page...

Les prochaines pages...

Définitions

Objet:	Un objet est une instance d'une classe (!). En fait, un objet est une "entité" informatique qu'un programme informatique peut manipuler. Dans le cadre du présent tutoriel, un objet est une entité externe à VBA que votre programme VBA peut (veut?) manipuler. Le cas le plus évident est une cellule de classeur. Pour la manipuler, VBA utilise une instance de la classe **Range**.
Classe:	Une classe est une catégorie (sorte) d'objets. Votre programme peut manipuler une cellule précise (ou un groupe de cellule), mais pas le concept de cellule.
Propriétés:	Chaque objet possède des propriétés qui le décrivent. La liste des propriétés varie d'une classe à l'autre, mais tous les objets d'une même classe ont les mêmes propriétés. Ce qui les distingue, ce sont les valeurs des propriétés. Par exemple, toutes les feuilles Excel ont la propriété **Name**, mais la valeur de la propriété **Name** est différente pour chaque feuille (du moins on le souhaite).
Méthode:	Une méthode est une action que peut réaliser un objet. La liste des méthodes varie d'une classe à l'autre, mais tous les objets d'une même classe ont les mêmes méthodes. Par exemple, la méthode **Copy** d'une cellule permet de copier le contenu de cette cellule dans le presse-papier (ou dans une autre cellule). Évidemment, il arrive qu'une méthode change les propriétés d'un objet. Par exemple, la méthode **PasteSpecial** d'une cellule change une ou plusieurs propriétés de cette cellule.
Événement:	Un événement permet d'associer une procédure VBA à un objet. Par exemple, l'événement **Change** d'une feuille se produit lorsqu'une cellule de cette feuille est modifiée et permet d'associer une procédure nommée `Worksheet_`**Change** à cette feuille Excel**.**
Collection:	Ensemble d'objets d'une même classe qui peuvent être adressés avec le nom de la collection et un numéro d'item. Par exemple, un classeur Excel est une collection **Sheets** d'objets **WorkSheet** et/ou **Chart**

Modèle Objet Excel

VBA peut manipuler le contenu d'un classeur Excel par les objets qu'il contient. Mais comment trouver le nom de l'objet (ou de la classe) qu'il faut utiliser?

Le plus simple est d'utiliser l'enregistreuse de macro-commande.

Par exemple, pour trouver comment changer la couleur de fonds d'une feuille Excel,

- Démarrez l'enregistreuse de macro-commandes, en notant l'endroit où elle sera enregistrée.

- Changez la couleur du fonds (sélectionner la feuille, afficher la boîte de dialogue Format de cellule, choisir l'onglet remplissage puis une couleur)
- Arrêtez l'enregistreuse de macro-commande
- Affichez-là dans l'éditeur VBA. Elle ressemble à:

```
Sheets("Feuil1").Select
Cells.Select
With Selection.Interior
    .Pattern = xlSolid
    .PatternColorIndex = xlAutomatic
    .ThemeColor = xlThemeColorDark2
    .TintAndShade = 0
    .PatternTintAndShade = 0
End With
```

  Dans cet enregistrement, on reconnait les objets:

  - Sheets("Feuil1"): la feuille Feuill1
  - Cells : collection de toutes les cellules de la feuille
  - Interior: collection de tous les intérieurs des cellules

  et les méthodes:

  - Select

- L'objet recherché est donc:
  Sheets("Feuil1").Cells.Interior
- L'aide VBA Excel nous indique alors que cet objet possède une propriété Color. On peut donc programmer:
  `Sheets("Feuil1").Cells.Interior.Color = vbYellow`
  Ce qui est beaucoup plus efficace que le code produit par l'enregistreuse.

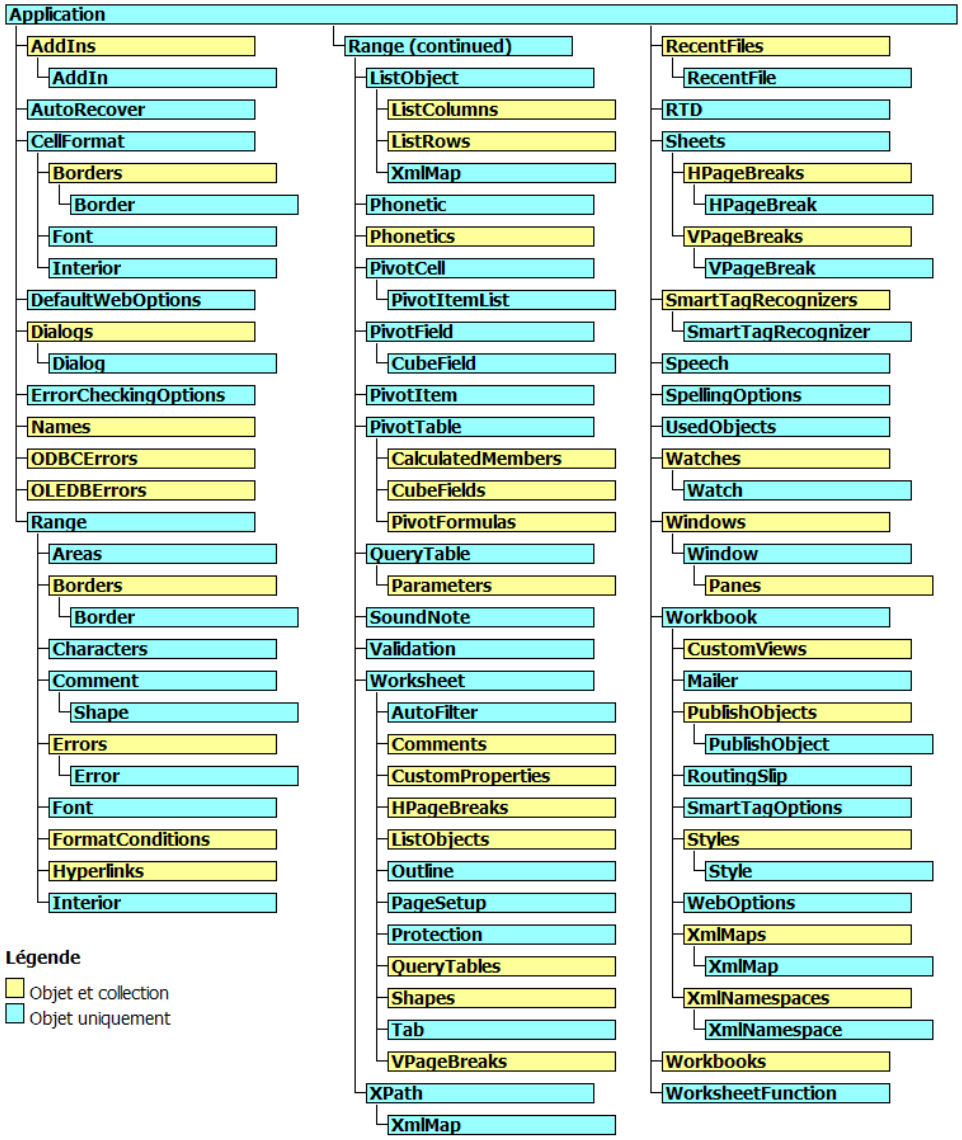
Une façon plus complexe est d'explorer le modèle objet Excel.
L'aide VBA Excel 2007 permet de l'explorer en accédant à la rubrique:
Référence du développeur Excel 2007
Microsoft vous offre ici la liste des principaux objets Excel en ordre alphabétique, ce qui est pratiquement inutilisable.

Le document suivant explique comment installer l'aide locale VBA:

Installer l'Aide VBA locale

Vous pouvez y parcourir la hiérarchie des objets Excel:

Modèle objet Microsoft Excel



et trouver l'objet qui vous intéresse en cliquant d'abord sur Range, puis en trouvant la propriété ou l'objet Interior, et en devinant qu'il faut adresser l'objet Interior de la collection Cells de la feuille voulue.

Principaux objets Excel

Les cellules Excel sont des objets de la classe **Range**.
La meilleure façon d'accéder aux propriétés d'une plage de cellules Excel depuis VBA est de nommer cette plage en Excel, puis d'utiliser ce nom comme référence de Range.

- **Range**("plage") est un objet contenant la plage nommée... plage.

Une technique beaucoup moins utile (mais hélas trop répandue) est d'utiliser l'adresse de la plage.

- **Range**("A1:H5") est un objet contenant la plage A1:H5.

Une variante de cette technique est l'utilisation de la méthode **Evaluate** (voir plus loin).

- **Evaluate**("A1:H5") ou mieux encore [A1:H5] équivalent exactement à Range("A1:H5").

Aussi peu utile.

Deux autres façons utilisent des numéros de ligne et de colonne, et sont beaucoup plus utiles.

- **Cells**(3,5) est un objet contenant la cellule E5 (ligne 3, colonne 5).

Attention, l'objet Cells utilise l'adressage Ligne, Colonne, alors que Excel utilise Colonne, Ligne (E5).

- **Range**("A1").**Offset**(3,5) est un objet contenant la plage F4 soit un déplacement de 3 lignes et 5 colonnes à partir de A1.

Ces deux dernières techniques peuvent être utilisées avec l'objet **Range** pour désigner des plages.

- **Range**(Cells(3,5), **Range**("A1").**Offset**(3,5)) est un objet contenant la plage E3:F4.

L'utilisation d'un nom de plage au lieu d'une adresse permettra de déplacer la plage dans Excel sans devoir modifier la programmation VBA.
Il est donc FORTEMENT suggéré de manipuler les lignes d'une liste de données ainsi:

- Nommez la cellule de gauche de l'entête de liste
- Utilisez la notation **Range**(*nom*).**Offset**(*ligne,colonne*) pour représenter les champs de la liste.

Cellule ou plage? Le modèle objet Excel ne fait pas la différence entre une plage et une cellule. Une cellule est simplement une plage ne contenant qu'une cellule.
Range("A1:B9") est un objet de la même classe que **Range**("A1") et possède donc les mêmes propriétés (dont les valeurs peuvent être différentes) et méthodes.

Un objet **Range** est aussi une collection de cellules, Comme une cellule est un Range (d'une cellule), c'est une collection ne comptant qu'un objet.
Un Range est donc une collection de collections.

Plages spéciales. Certaines propriétés du modèle objet Excel contiennent des plages spécifiques:

Cells	Plage qui contient toutes les cellules d'une feuille: • **ActiveSheet**.**Cells** • **Sheets**("Province").Cells On peut adresser une cellule de cette plage: **Sheets**("Province").Cells(3,5) désigne la cellule E3 de la feuille Province.
UsedRange	Plage délimitée par la première (haut-gauche) et la dernière (bas-droite) des cellules occupées d'une feuille: • **ActiveSheet**.**UsedRange** • **Sheets**("Province").**UsedRange**
CurrentRegion	Plage continue (limitée par des lignes et des colonnes vides) dont fait partie la cellule: • **Range**("Titre").**CurrentRegion**
Rows	Collection des lignes d'une plage. Permet d'obtenir une ligne d'une plage: • **Range**("ListeProvinces").**Rows**(1) • **Range**("Titre").**CurrentRegion**.**Rows**(1)
Columns	Collection des colonnes d'une plage. Permet d'obtenir une colonne d'une plage: • **Range**("ListeProvinces").**Columns**(1) • **Range**("Titre").**CurrentRegion**.**Columns**(1)
SpecialCells	Plages spéciales d'une plage Excel: • **SpecialCells**(xlCellTypeBlanks) Cellules vides • **SpecialCells**(xlCellTypeConstants). Cellules contenant des constantes • **SpecialCells**(xlCellTypeFormulas). Cellules contenant des formules • **SpecialCells**(xlCellTypeLastCell). Dernière cellule dans la plage utilisée • **SpecialCells**(xlCellTypeVisible).Toutes les cellules visibles • ...

Les objets Range ont un grand nombre de propriétés. Voici les plus intéressantes:

Column	Numéro de la colonne de gauche de la plage • **Range**("B4:H6").**Column** vaut 2
Columns.Count	Nombre de colonnes de la plage • **Range**("B4:H6").**Columns**.Count vaut 7
Row	Numéro de la ligne du haut de la plage
Rows.Count	Nombre de lignes de la plage
Count	Nombre de cellules de la plage
CurrentRegion	Plage continue (limitée par des lignes et des colonnes vides) dont fait partie la cellule.
End(xlDown)	Cellule de la dernière ligne à la fin de la zone qui contient la plage. Correspond aux touches Fin+Bas.
End(xlToRight)	Cellule de la dernière colonne à la fin de la zone qui contient la plage. Correspond aux touches Fin+Droite.

Text	Valeur AFFICHÉE de la cellule, donc évaluée et formatée.
Value	Valeur ÉVALUÉE de la cellule.
Worksheet	Objet Worksheet qui est la feuille qui contient la plage

Les feuilles Excel sont des objets de la classe **Worksheet**.

Elles sont regroupées dans la collection **Worksheets** et dans la collection Sheets.

On peut manipuler une feuille de classeur par son nom:

- **Worksheets**("Provinces") est la feuille nommée province.
- **Sheets**("Provinces") est aussi la feuille nommée province.

La différence entre les deux collections est que la collection **Sheets** contient aussi les objets **Chart** du classeur.

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

Tous les classeurs ouverts sont regroupés dans la collection **Workbooks**.

- **Workbook**("Exemples Objets.xlsm") est le classeur Exemples Objets.xlsm

La programme Excel lui-même, lorsque démarré, est un objet de la classe **Application**.

Le classeur actif est:
Application.ActiveWorkbook ou plus simplement **ActiveWorkbook**.

Le classeur contenant le code VBA en cours d'exécution est:
Application.ThisWorkbook ou plus simplement **ThisWorkbook**.

La feuille active du classeur actif est:
Application.ActiveWorkbook.ActiveSheet ou plus simplement **ActiveSheet**.

La cellule active de la feuille active du classeur actif est:
Application.ActiveWorkbook.ActiveSheet.ActiveCell ou plus simplement **ActiveCell**.
La cellule active de la feuille nommée Province du classeur actif est:
Application.ActiveWorkbook.Worksheets("Province").ActiveCell ou plus simplement **Worksheets("Province").ActiveCell.**
etc.

L'objet **Selection** est l'objet passe-partout, qui contient ce qui est sélectionné dans le classeur Excel. Ce peut être une plage, un graphique, un tableau croisé ou n'importe quoi. L'utilisation de **Selection** en VBA est à éviter, puisqu'il risque fort de contenir une sélection inattendue lors de l'exécution du programme. Si vous utilisez l'objet Selection, n'oubliez pas de valider le type de l'objet qu'il contient.

La barre d'état de la fenêtre Excel est l'objet **StatusBar**.

Méthodes Excel

La plupart des opérations que l'utilisateur peut faire en Excel peuvent être programmées en **VBA** grâce aux méthodes des différents objets du modèle objet Excel.

Voici les plus intéressantes:

Evaluate	Permet d'exécuter en VBA une formule Excel (ce qui est à droite du signe = dans une cellule). La méthode Evaluate peut être remplacée par les crochets: [2+2] équivaut à Evaluate("2+2") et vaut 4. Attention, ce qui est incorrect dans une cellule est incorrect pour Evaluate. • [A1] équivaut à **Range**("A1").Value • [Sum("A2:H6")] exécute la formule Excel Somme("A2:H6")
Intersect	Retourne une plage formée des cellules communes de deux plages. • **Intersect**(**Range**("A1:C4"),**Range**("B2:D5")) équivaut à **Range**("B2:C4") Cette méthode permet de savoir si une cellule appartient à une plage: • (**Intersect**(**Range**("B6"),**Range**("Provinces") **is Nothing**) est faux si B6 fait partie de la plage Provinces.
Calculate	Recalcule la feuille ou le classeur Excel • **Calculate** • **ActiveWorksheet**.**Calculate**
Save	Enregistre le classeur • **ActiveWorkbook**.**Save**
Quit	Ferme Excel. Attention, si le classeur a été modifié mais non sauvegardé, la boîte de dialogue de sauvegarde d'Excel sera affichée. Voir la méthode Save. • **Application**.**Quit**
Clear **ClearContents** **ClearFormats**	Effacent respectivement les cellules, les contenus ou les formats d'une plage. • **Range**("A1").**Clear** • **Worksheet**("Tableaux").**Clear**

Copy	Copie le contenu d'une plage dans une autre ou dans le presse-papier • **Range**("A1").**Copy Range**("B1") copie le contenu de A1 dans B1 • **Range**("A1").**Copy** copie le contenu de A1 dans le presse-papier>
PasteSpecial	Colle le contenu d'une plage ou du presse-papier dans une plage. On peut spécifier le type de collage spécial: xlPasteAll (défaut), xlPasteAllExceptBorders, xlPasteColumnWidths. xlPasteComments, xlPasteFormats, xlPasteFormulas, xlPasteFormulasAndNumberFormats, xlPasteValidation, xlPasteValues ou xlPasteValuesAndNumberFormats. • **Range**("Total").**Copy** copie la plage Total dans le presse-papier • **Range**("A1").PasteSpecial (xlPasteValues) copie la VALEUR du presse-papier dans la cellule A1. li>
Find et **FindNext**	Ces deux méthodes permettent d'effectuer une recherche dans une plage de cellules. Voir Exemple 6.
Select	Sélectionne une cellule ou une feuille Excel. À éviter la plupart du temps en programmation **VBA**, car le déplacement du curseur Excel est rarement souhaitable pendant l'exécution d'un programme **VBA**.
Sort	Tri de plage. Voir l'aide VBA Excel pour en connaître le fonctionnement. Il est aussi utile d'utiliser l'enregistreuse de macro-commandes sur un tri manuel des données pour en analyser les paramètres.
Volatile	Force l'exécution d'une Function à chaque fois que le classeur est recalculé. • **Application**.**Volatile** Normalement, Excel recalcule les formules des cellules lorsque leurs antécédants sont modifiés. Dans le cas des fonctions personnalisées VBA (Function) utilisées dans Excel, le recalcul ne se fait pas toujours. Application.Volatile utilisée dans une procédure Function assure que la Function sera réexécutéer à chaque recalcul du classeur Excel.

Principaux événements Excel

Objet	Événement	Paramètres	Se produit lorsque	Utilisé pour
Workbook	**Open**		Le classeur Excel est ouvert	• Afficher un écran d'accueil • Afficher ou masquer des éléments d'Excel ou du classeur • Initialiser des paramètres • ...
Workbook	**BeforeClose**	**Cancel**	Lorsqu'une demande de fermeture du classeur a été faite	• Contrôler ou empêcher la sauvegarde du classeur • Gérer des copies de sécurité • Contrôler ou empêcher la fermeture du classeur • Rétablir l'affichage d'éléments Excel • ...
Workbook	**SheetChange**	**Sh** la feuille modifiée **Target** la plage modifiée	Lorsqu'une cellule ou plage du classeur est modifiée	• Contrôler ou empêcher la modification d'une cellule (voir exemples). Utiliser lorsque la même procédure s'applique à toutes les plages du classeur
Worksheet	**Change**	**Target** la plage modifiée	Lorsqu'une cellule ou plage de la feuille qui contient la procédure Sub est modifiée	• Contrôler ou empêcher la modification d'une cellule (voir exemples). Utiliser lorsque la procédure ne s'applique qu'aux plages d'une seule feuille
Worksheet	**SelectionChange**	**Target** la nouvelle plage sélectionnée	Lors de la sélection d'une autre plage dans un classeur	• Contrôler ou empêcher le déplacement du curseur dans un classeur Excel (voir exemples).

Remarquez que dans les événements **SheetChange**, **Change** et **SelectionChange**, **VBA** n'a pas directement accès aux valeurs précédant le changement, ni à un paramètre Cancel pour annuler le changement. Il faut utiliser une astuce pour empêcher le changement. Voir les exemples.

Notez finalement qu'on peut désactiver le traitement des événements, en utilisant la propriété **EnableEvents**.

Autres objets intéressants

Me

C'est un objet Excel ou VBA contenant le code en cours d'exécution. Si le code appartient à une feuille, c'est un objet Worksheet contenant cette feuille. Me est le plus souvent utilisé pour désigner le formulaire VBA (Forms) contenant le code en cours d'exécution. On évite ainsi de nommer le formulaire dans le code VBA , ce qui donne des programmes plus faciles à utiliser dans d'autres formulaires.

- **Me.Name** contient le nom du formulaire en cours d'exécution.

Utilisation la plus fréquente:

- **Me.Close** ferme le formulaire qui contient cette ligne.

Err

C'est un objet VBA contenant la dernière erreur rencontrée. Ses propriétés permettent d'en connaître la nature:

- **Err.Description** contient la description de la dernière erreur rencontrées par VBA.

VBA peut déclencher une erreur avec Err:

- **Err.Raise 18** simule une interruption par l'utilisateur (Ctrl-Break).

FileSystemObject

C'est l'objet contenant des instructions permettant à **VBA** d'accéder à l'arborescence de dossiers et de fichiers de l'ordinateur.

ATTENTION ce n'est pas un objet **VBA** ni un objet Excel, mais un objet de la librairie Microsoft Scripting Runtime.

Vous devez référencer cette librairie dans l'éditeur de code **VBA** avant de l'utiliser. Voir Exemples.

Objets intéressants:

Drives	Collection d'objets Drive
Drive	Objet contenant un lecteur
Folders	Collection d'objets Folder
Folder	Objet contenant un dossier
Files	Collection d'objets File
File	Objet contenant un fichier

Un objet **FileSystemObject** contient une collection **Drives**.
Chaque objet **Drive** de la collection **Drives** contient une collection **Folders** et une collection **Files**.
Chaque objet **Folder** d'une collection **Folders** contient une collection **Folders** et une collection **Files**.
Chaque objet **File** d'une collection **Files** contient un fichier (document **Windows**).

Cette structure permet de parcourir l'arborescence de documents en **VBA** comme avec l'explorateur **Windows**.

Autres objets, autres propriétés

Les environnements Excel, Office et Windows offrent une richesse incroyable d'objets et de propriétés qu'il est impossible de décrire ici
(ni ailleurs d'ailleurs), ne serait-ce que parce que cette liste change constamment

Il vous faudra donc développer l'habilité de soupçonner l'existence de l'objet que vous souhaitez utiliser et d'en découvrir le nom et la documentation.
Bonne chance.

Objet implicite (With)

Il arrive fréquemment qu'une série d'instructions utilise le même objet. L'écriture (et la lecture) du code **VBA** devient alors fastidieuse, avec la répétition incessante de la même référence.

L'instruction With permet de réduire les répétitions.

With *objet*
  [*instructions*]
End With

Où:

- *objet* Nom d'un objet.
- *instructions* Une ou plusieurs instructions VBA

À l'intérieur du bloc With ... End With, toute référence commençant par un . (point) appartient à *objet*.

ATTENTION: With N'est **PAS** une instruction de boucle ou de répétition ou de test. Si le code doit être répété, il doit être encadré d'une instruction de boucle (Do) ou de test (If).

Dans l'exemple 2 ci-dessous, le code:

```
    If Target.Interior.Color = Range("CouleurVerrou").Interior.Color Then
        Application.EnableEvents = False 'Empêcher une boucle d'événements
        Application.Undo                 'Annuler le changement
        Application.EnableEvents = True  'Très important de réactiver
    End If
```

pourrait être:

```
    With Application
        If Target.Interior.Color = Range("CouleurVerrou").Interior.Color Then
            .EnableEvents = False 'Empêcher une boucle d'événements
            .Undo                 'Annuler le changement
            .EnableEvents = True  'Très important de réactiver
        End If
    End With
```

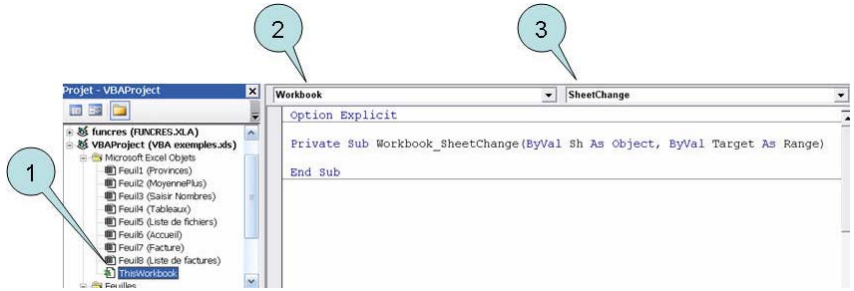
Exemples

Exemple 1: Changer la couleur d'arrière-plan des cellules modifiées.

- Créez la procédure événementielle

  1. Double-cliquez sur ThisWorkbook pour ouvrir le module VBA associé au classeur
  2. Choisissez Workbook dans la liste d'objets
  3. Choisissez SheetChange dans la liste d'événements

  La procédure Sub Workbook_SheetChange est créée pour vous.



- Insérez le code ci-contre

```
Private Sub Workbook_SheetChange(ByVal Sh As Object, ByVal Target As Range)
'Auteur: Michel Berthiaume
'Changer l'arrière-plan d'une cellule modifiée

    'Target est la plage modifiée (voir les paramètres de la procédure, ci-dessus)
    Target.Interior.Color = vbYellow 'Jaune

End Sub
```

- Testez le programme en Excel: toute cellule modifiée devient jaune.

Il est possible de restreindre les modifications à une seule feuille:

- en insérant le code dans la procédure événementielle Worksheet_Change du module associé à la feuille voulue.
  ou
- en insérant un test sur le nom de la feuille (Sh.Name) dans le code suggéré (en le laissant dans le module associé au classeur).

Si les deux techniques fonctionnent, la seconde limite l'éparpillement du code dans plusieurs modules et produit donc un programme plus facile à comprendre et à modifier plus tard.

Exemple 2: Une fonction qui affiche le contenu réel d'une cellule.

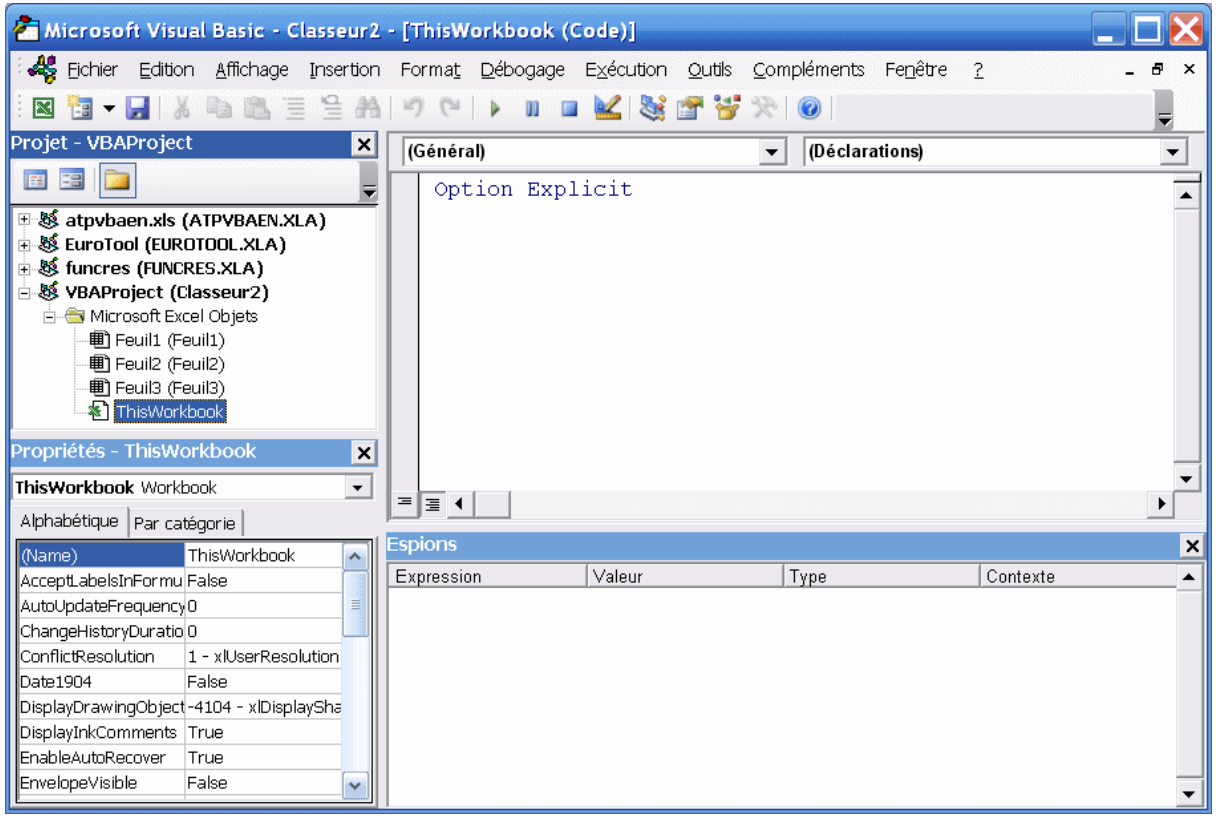
- Dans un module VBA Excel, insérer le code ci-joint:Ouvrez le module VBA associé au classeur Excel

```
Function fnValeurCellule(rCellule)
'Auteur: Michel Berthiaume
'Retourne le contenu (formule ou valeur) de la cellule en paramètre
    If TypeName(rCellule) <> "Range" Then
        fnValeurCellule = "Erreur de paramètre"
        Exit Function
    End If
    fnValeurCellule = rCellule.Formula
End Function
```

Dans une (ou plusieurs) cellules du classeur Excel, utilisez la fonction
=fnValeurCellule(A2)
en remplaçant A2 par une référence à une cellule Excel dont vous voulez afficher le contenu réel.

Exemple 3: Empêcher la modification des cellules dont l'arrière-plan est rouge.

Ouvrez le module VBA associé au classeur Excel



- Insérez le code ci-contre

```
Private Sub Workbook_SheetChange(ByVal Sh As Object, ByVal Target As Range)
'Auteur: Michel Berthiaume
'Empêcher la modification de cellules dont l'arrière-plan
' est d'une couleur choisie par l'utilisateur
'Note: la feuille doit contenir une cellule nommée CouleurVerrou
' de la couleur choisie

'Target est la plage modifiée (voir les paramètres de la procédure, ci-dessus)
    If Target.Interior.Color = Range("CouleurVerrou").Interior.Color Then
        Application.EnableEvents = False 'Empêcher une boucle d'événements
        Application.Undo                 'Annuler le changement
        Application.EnableEvents = True  'Très important de réactiver
    End If
End Sub
```

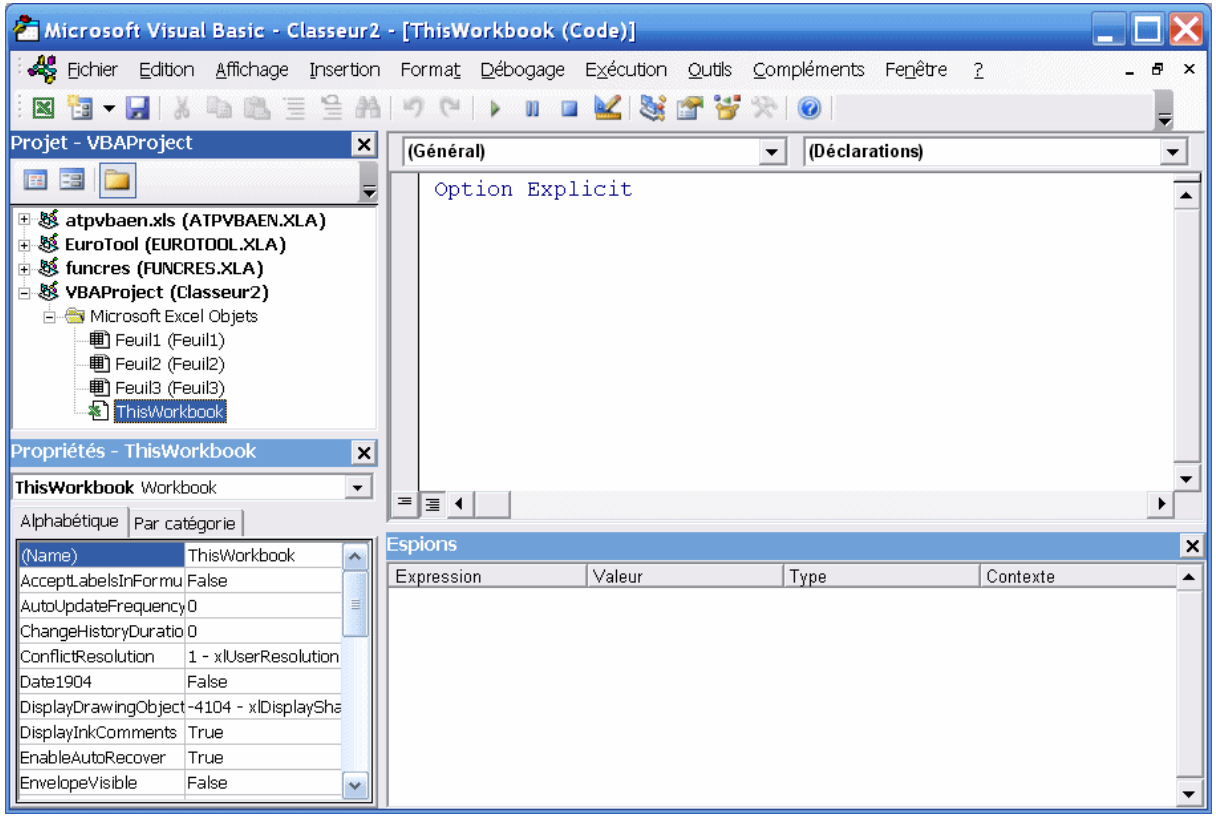
- Dans Excel, changez l'arrière-plan d'une plage de cellules.
- Créez une plage nommée CouleurVerrou à partir d'une cellule de la plage ainsi colorée.
- Testez le programme en Excel: toute changement à une cellule de la plage colorée est annulée par le programme

---

Remarquez ici l'utilisation de la propriété EnableEvents pour empêcher que l'instruction suivante (la méthode Undo) provoque l'événement Workbook_SheetChange et entraîne ainsi une succession sans fin d'exécution de la procédure Sub correspondante.
Workbook_SheetChange appelant Workbook_SheetChange appelant Workbook_SheetChange appelant Workbook_SheetChange ...

Exemple 4: Empêcher la modification des cellules ne faisant pas partie d'une plage.

- Ouvrez le module VBA associé au classeur Excel



- Insérez le code ci-contre

```
Private Sub Workbook_SheetChange(ByVal Sh As Object, ByVal Target As Range)
'Auteur: Michel Berthiaume
'Empêcher la modification de cellules en dehors
' d'une plage nommée ZoneDeSaisie
'Note: le classeur doit contenir une plage nommée ZoneDeSaisie

    'Target est la plage modifiée (voir les paramètres de la procédure, ci-dessus)
    If Intersect(Target, Range("ZoneDeSaisie")) Is Nothing Then
        Application.EnableEvents = False  'Empêcher une boucle d'événements
        Application.Undo                  'Annuler le changement
        Application.EnableEvents = True   'Très important de réactiver
    End If
End Sub
```

- Dans Excel, créez une plage nommée ZoneDeSaisie.
- Testez le programme en Excel: toute changement à une cellule ne faisant pas partie de la plage ZoneDeSaisie est annulée par le programme.

Exemple 5: Empêcher le déplacement du curseur Excel tant qu'une condition n'est pas respectée.

L'événement qui nous intéresse est **SelectionChange**, qui n'a pas de paramètre Cancel, donc qu'on ne peut pas annuler directement.
La méthode Undo ne fonctionne pas non plus, car elle n'annule pas le déplacement. De plus, elle risquerait d'annuler une modification, alors qu'on ne veut annuler que le déplacement.
Il faudra donc que le programme garde trace de la dernière cellule sélectionnée.

Dans le programme ci-dessous, dès que l'utilisateur sélectionne une cellule de la plage ZoneDeSaisie, il ne peut pas déplacer le curseur hors de la plage tant que la somme des cellules de la plage sera différente de 10.

- Insérez les fonctions ci-contre dans un module VBA:

```
Function fnDéplacementValide()
'Auteur: Michel Berthiaume
'Retourne Vrai si la somme des cellules de la plage ZoneDeSaisie est égale à 10

    fnDéplacementValide = ([sum(ZoneDeSaisie)] = 10)

End Function
Function fnPlageAdansPlageB(plageA, PlageB)
'Auteur: Michel Berthiaume
'Retourne Vrai si plageA est complètement dans PlageB ou l'inverse

    If Intersect(plageA, PlageB) Is Nothing Then
        fnPlageAdansPlageB = False
```

```
    Else
        fnPlageAdansPlageB = (Intersect(plageA, PlageB).Count = plageA.Count)
    End If

End Function
```

- Au début du module VBA ThisWorkbook sous **Option Explicit**, insérez

```
Public rCellulePrécédente As Range    'Contiendra la dernière cellule sélectionnée
```

- Dans la procédure Sub Workbook_Open() insérez:

```
Set rCellulePrécédente = ActiveCell   'Initialiser à la sélection initiale (à l'ouverture)
```

- Dans la procédure Sub Workbook_SheetSelectionChange() insérez:

```

```

Exemple 6: Trouver toutes les cellules d'une feuille contenant le mot Liberté.

Technique 1, utilisant une boucle testant chaque cellule d'une plage. Notez bien que le résultat est une plage de cellules disjointes, chacune contenant le mot cherché.
Notez aussi qu'on a généralisé la solution, qui permet de chercher n'importe quelle chaîne de caractères dans n'importe quelle plage Excel.

- Insérez les fonctions ci-contre dans un module VBA:

```
Function fnTrouveMotDansPlage(sMot, rPlage)
'Auteur: Michel Berthiaume
'Retourne une plage formée des cellules de rPlage contenant sMot
Dim rTrouvée As Range 'contiendra les cellules trouvées
Dim rCellule As Range 'contiendra chaque cellule testée, à tour de rôle

    For Each rCellule In rPlage    'Pour chaque cellule de rPlage
        'Si sMot est dans rCellule
        If InStr(UCase(rCellule.Value), UCase(sMot)) > 0 Then
            If rTrouvée Is Nothing Then  'Première cellule trouvée
                Set rTrouvée = rCellule
            Else                         'Cellules trouvées suivantes
                Set rTrouvée = Union(rTrouvée, rCellule)
            End If
        End If
    Next
    Set fnTrouveMotDansPlage = rTrouvée
End Function
```

- Le nombre de cellules contenant le mot Liberté trouvées dans la plage A1:C6 peut maintenant être obtenu avec l'expression VBA:

```
fnTrouveMotDansPlage("Liberté", Range("A1:C6")).Count
```

- ou avec la formule Excel

```
=NBVAL(fnTrouveMotDansPlage("Liberté";A1:C6))
```

- On peut mettre ces cellules en rouge avec l'instruction VBA:

```
fnTrouveMotDansPlage("Liberté", Range("A1:C6")).interior.color = vbRed
```

Remarques sur cette fonction

- Puisqu'elle retourne une plage de cellules, elle ne peut pas être utilisée directement dans une cellule du classeur Excel. En effet, une cellule ne peut pas contenir une plage.
  Mais on peut l'utiliser comme paramètre de type plage dans une fonction Excel, comme dans
  =NBVAL(fnTrouveMotDansPlage("Liberté";A1:C6))
- La fontion Excel Nb.Si() joue un rôle semblable, mais dans sa version simple (=NB.SI(A1:C6;"Liberté")) ne comptera que les cellules dont la valeur est **égale** à la valeur cherchée, contrairement à fnTrouveMotDansPlage qui trouve les cellules qui **contiennent** la valeur. De plus, fnTrouveMotDansPlage retourne la plage et non le nombre. On peut donc manipuler le résultat en VBA en lui assignant un nom de variable:
  ```
  dim rPlageTrouvée as Range
  Set rPlageTrouvée = fnTrouveMotDansPlage("Liberté", Range("A1:C6"))
  ```

Technique 2, utilisant les méthodes Find et Findnext dans une boucle cherchant les cellules d'une plage. Notez bien que le résultat est le même que dans la technique 1.

- Insérez les fonctions ci-contre dans un module VBA:

```
Function fnTrouveMotDansPlage(sMot, rPlage)
'Auteur: Michel Berthiaume
'Retourne une plage formée des cellules de rPlage contenant sMot

Dim rTrouvées As Range
Dim rCelluleTrouvée As Range
Dim rPremièreTrouvée As Range

    Set rCelluleTrouvée = rPlage.Find(sMot)
    If rCelluleTrouvée Is Nothing Then
        Exit Function 'Aucune occurrence
    End If

    'Find et FindNext trouveront toutes les occurrences,
    'puis recommenceront à la première
    'Il faut donc chercher jusqu'à ce qu'on trouve la première à nouveau.
    Set rPremièreTrouvée = rCelluleTrouvée
    Set rTrouvées = rCelluleTrouvée
    Set rCelluleTrouvée = rPlage.FindNext(rCelluleTrouvée)

    Do While Not rCelluleTrouvée = rPremièreTrouvée
        Set rTrouvées = Union(rTrouvées, rCelluleTrouvée)
        Set rCelluleTrouvée = rPlage.FindNext(rCelluleTrouvée)
    Loop
    Set fnTrouveMotDansPlage2 = rTrouvées
End Function
```

Exemple 7: Obtenir le nom et la taille du dossier contenant le classeur Excel actif.

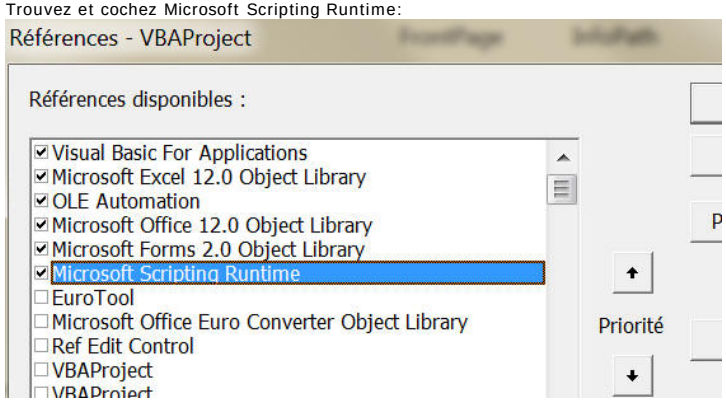
- Insérez les fonctions ci-contre dans un module VBA:

```
Function fnDossier(Optional param = 0)
'Auteur Michel Berthiaume
'Retourne le nom du dossier et/ou l'espace qu'il utilise

'La ligne suivante requiert une référence à Microsoft Scripting Runtime
Dim fso As New Scripting.FileSystemObject

    If param = 1 Then
        fnDossier = ActiveWorkbook.Path
        Exit Function
    ElseIf param = 2 Then
        fnDossier = (fso.GetFolder(ActiveWorkbook.Path).Size) / 1024
        Exit Function
    Else
        fnDossier = ActiveWorkbook.Path & ": " & _
            Round(fso.GetFolder(ActiveWorkbook.Path).Size / 1024, 2) & _
            " Ko"
    End If

End Function
```

Remarques sur cette fonction

- Une fonction ne peut retourner qu'une seule valeur, alors qu'ici on veut le nom te la taille du dossier. On utilise donc un paramètre pour spécifier la valeur voulue.
  - 1 retourne le nom
  - 2 retourne la taille

  et si le paramètre est omis, la fonction retourne les deux dans une chaîne de caractères.
- Cette fonction utilise une librairie d'objets qui n'est pas inclue par défaut dans l'éditeur VBE. Il faut donc ajouter une référence à cette libraire:
  Dans VBE:
  - Ouvrez le menu Outils
  - Choisissez Références
  - Trouvez et cochez Microsoft Scripting Runtime:



Sommaire

Fichier exemples

Suite: Conseils de programmation

VBA - Conseils de programmation

Auteur: Michel Berthiaume

Les prochaines pages...

Liste d'instructions

Un bon programme:

- Fait **toujours** ce qui est prévu.
- Ne fait **jamais** ce qui n'est pas prévu.
- Est facile à **utiliser**.
- Est facile (peu coûteux) à **modifier**.

Pour vous aider à atteindre cet idéal, cette page vous offre quelques conseils.

- Si vous enregistrez une macro-commande, utilisez **l'aide VBA - Excel** (F1) sur chacun des éléments enregistrés. Bonne façon de se familiariser avec le modèle objet Excel.
- La 1ère instruction d'un module doit **TOUJOURS** être **Option Explicit.**
- Un code bien présenté a plus de chances de fonctionner:
  - Utilisez les retraits appropriés pour les instructions de test et de boucle
  - Préfixez toujours vos noms de variable pour éviter la confusion avec des mots réservés
  - Documentez: auteur, but de la procédure Sub ou fonction, astuces principales
  - Utilisez des noms de procédure Sub ou fonction qui décrivent ce qu'elles font
- Mettez toutes vos procédures Sub et fonctions dans un même module (nommé avec un nom descriptif)
- Documentez les étapes de vos programmes **AVANT** de les programmer
- Testez chaque ligne de code immédiatement après l'avoir écrite. Il est bien plus facile de trouver une erreur dans une ligne que 20 erreurs dans 20 lignes.
  Il est bien plus facile de commencer avec un programme qui fonctionne à moitié et ajouter une instruction à la fois que de déboguer un programme qui ne fonctionne pas du tout.
- Attention aux échéances: un programme qui fait la moitié de ce qui est prévu et le fait bien (à temps) est souvent bien plus apprécié qu'un programme qui ne fait rien du tout (mais sera prêt bientôt, promis promis)
- N'utilisez pas de variables de type variant, sauf pour les les paramètres de procédures
  - Validez le type des paramètres de procédures
  - Utilisez de préférence les types Currency, String et Date
  - Évitez les types Integer, Single, Object et Variant, qui peuvent causer des erreurs dans des situations limites
  - Évitez les variables globales; utiliser plutôt des cellules Excel ou des champs de formulaire
- Ne modifiez pas les valeurs des paramètres d'une procédure Sub ou Function
- Ne modifiez pas les objets Excel dans une **fonction**
- N'utilisez pas SELECT CASE, utilisez plutôt IF...THEN...ELSEIF...THEN...ELSE...END IF
- N'utilisez pas WHILE/WEND, ni SWITCH
- Évitez de sélectionner des cellules inutilement
- Utilisez i, j comme index pour des tableaux VBA, lLigne, lColonne pour des cellules Excel; de type Currency (long à la rigueur), JAMAIS Integer
- Conserver les paramètres et autres valeurs pouvant être modifiées par l'utilisateur dans Excel, pas dans VBA
- Une procédure fonction devrait toujours contenir une routine de gestion d'erreur (**On Error**)
- Lorsqu'on programme une division TOUJOURS tester que le diviseur est différent de 0

Erreurs fréquentes:

- Oublier d'utiliser Set pour assigner un objet à une variable
- Oublier une référence à une librairie dont on utilise les objets (classes)
- Division par 0
- Type erroné de paramètre

Suite (et fin) : Liste d'instructions

VBA - Liste (partielle) d'instructions VBA

Auteur: Michel Berthiaume

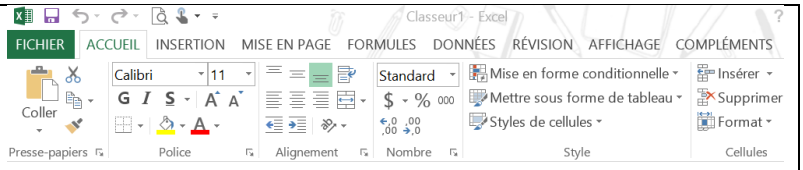
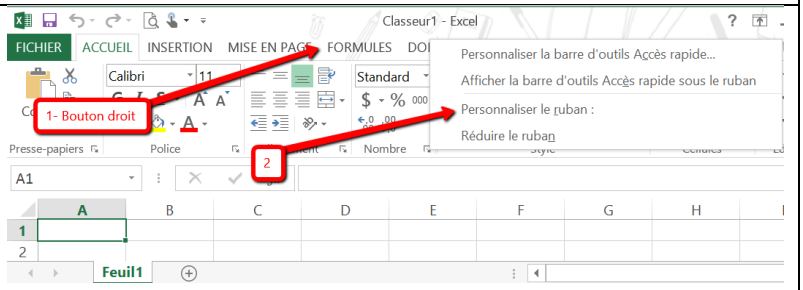
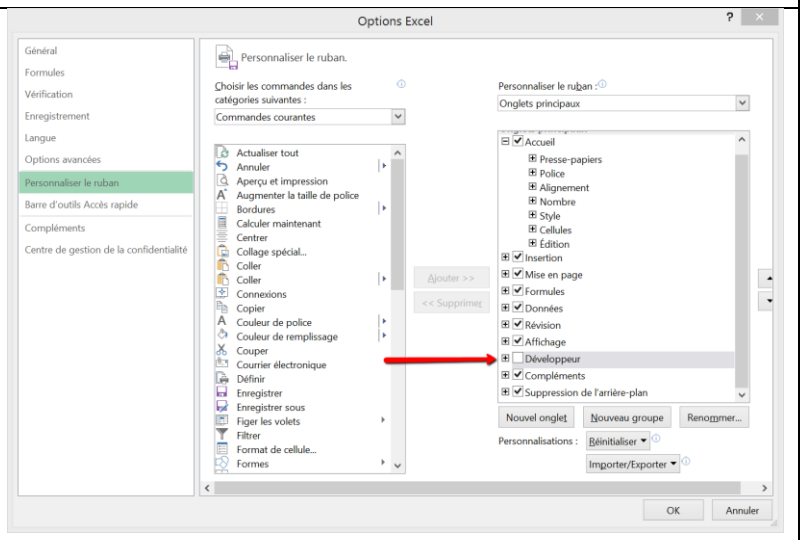
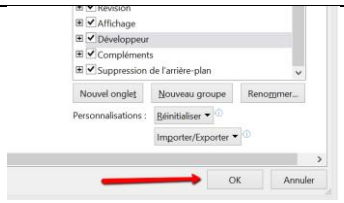
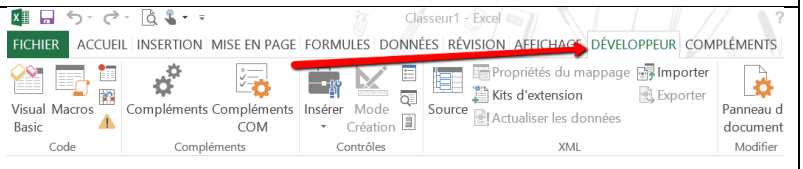
Vous trouverez ci-dessous des liens aux instructions VBA expliquées dans le site. Et le nom de quelques autres instructions utiles.

- Beep
- Call
- CBool()
- CByte()
- CCur()
- CDate()
- CDbl()
- CDec()
- Choose()
- CInt()
- CLng()
- Const
- Currency
- CSng()
- CStr()
- CVar()
- Date
- Date()
- Dim
- Do ... Loop
- DoEvents
- Double
- Err
- Exit
- For ... Next
- For Each ... Next
- Format()
- FormatCurrency
- FormatDateTime
- FormatNumber
- Function ... End Function
- GetSetting
- GoTo
- If ... End If
- IIf()
- InStr()
- Int()
- InputBox()
- Integer
- IsArray()
- IsDate()
- IsEmpty()
- IsError()
- IsMissing()
- IsNothing()
- IsNull()
- IsNumeric()
- IsObject()
- LBound()
- LCase()
- Left()
- Len()
-

- Let
- Long
- Me
- Mid()
- MsgBox()
- New
- Nothing
- Now()
- Null
- On Error
- Option explicit
- Private
- Public
- ReDim
- Replace
- Resume
- Right()
- Round()
- Select Case ... End Case
- Set
- SaveSetting
- Single
- Stop
- Str()
- String
- Sub ... End Sub
- Switch()
- Time()
- Trim()
- TypeName()
- TypeOf()
- UBound()
- UCase()
- Unload
- Val()
- Variant
- With ... End With

Configurer Excel : afficher l'onglet DÉVELOPPEUR

L'installation par défaut d'Excel ne donne pas facilement accès aux outils de développement VBA.

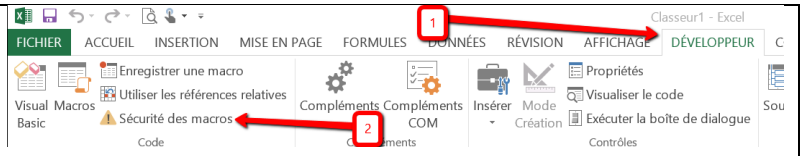
1	Démarrer Excel.	
	Si le ruban n'affiche pas l'onglet DÉVELOPPEUR : Cliquer avec le bouton droit sur un onglet Cliquer Personnaliser le Ruban	
2	Dans la boîte de dialogue Options Excel, cocher la case Développeur de la liste de droite. Il se peut que vous deviez faire défiler la liste pour afficher la case.	
3	Cliquer ensuite sur Ok	
	L'onglet DÉVELOPPEUR est maintenant disponible	

Configurer Excel : activer les macros

Il est possible que les macros soient désactivées dans la configuration d'Excel.

1 Dans l'onglet DÉVELOPPEUR

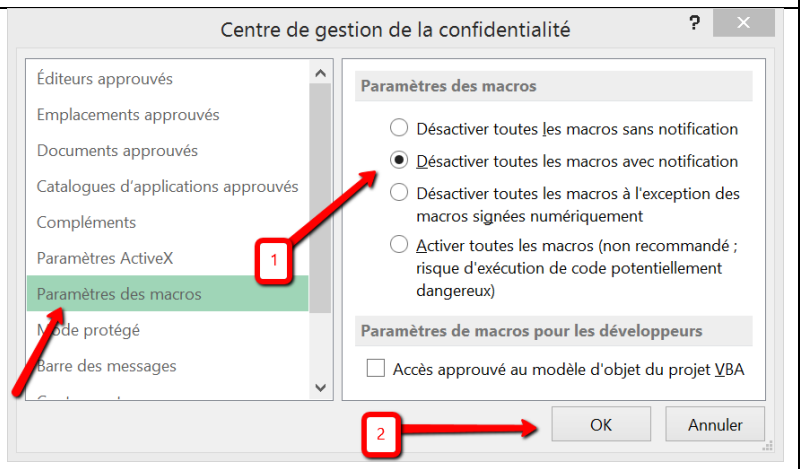
Cliquer Sécurité des macros



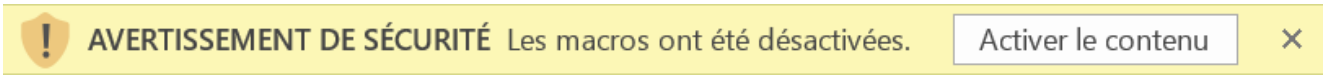
2 Dans le Centre de gestion de la confidentialité, l'onglet Paramètres des macros est activé.

Choisir Désactiver toutes les macros avec notification

Cliquer Ok



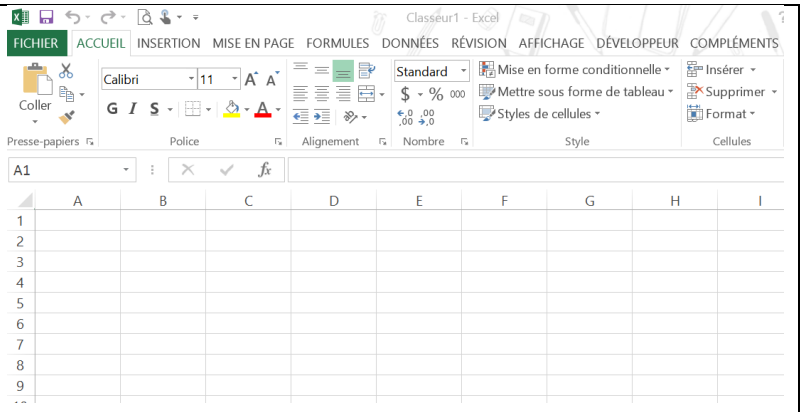
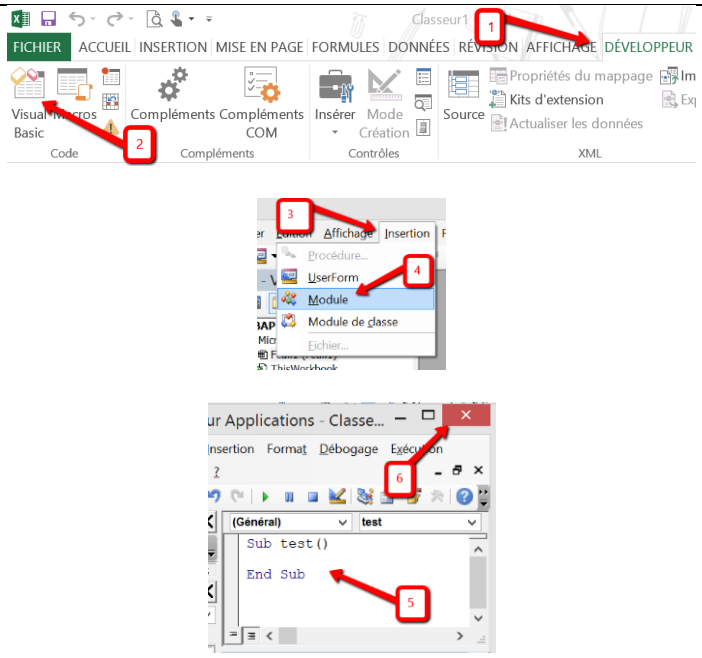
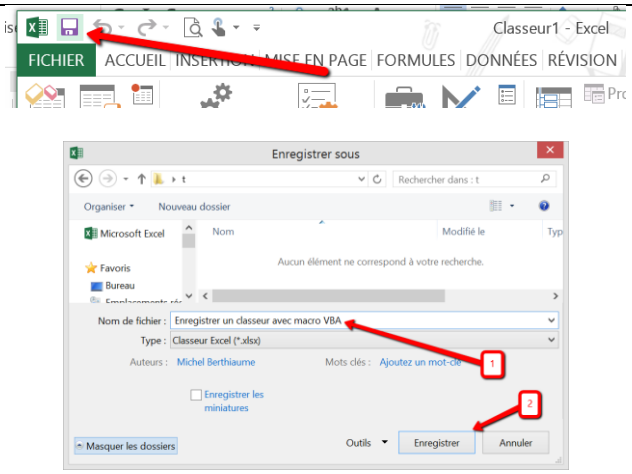
À l'ouverture d'un classeur Excel contenant des macros, l'avertissement suivant :

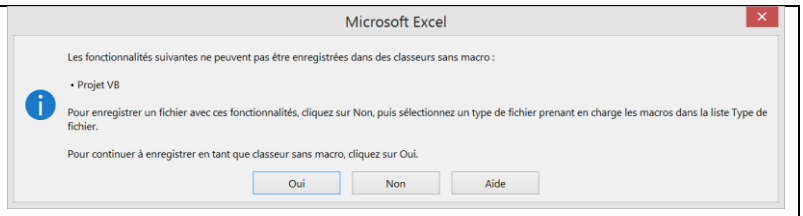
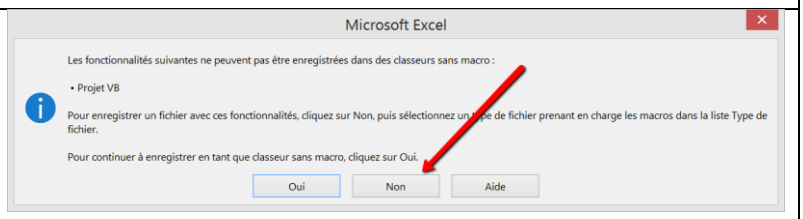
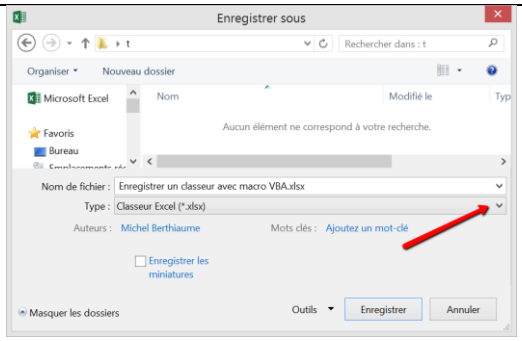
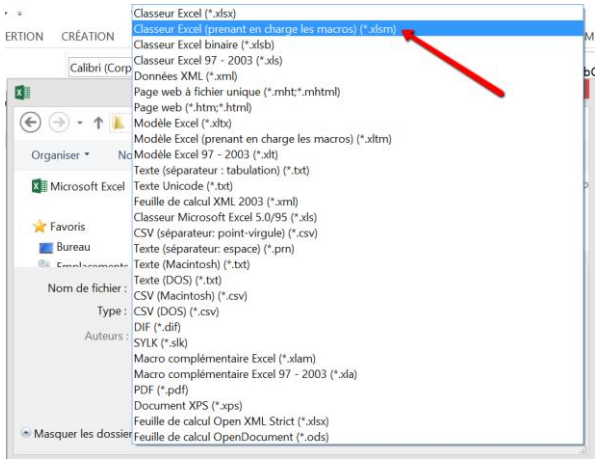


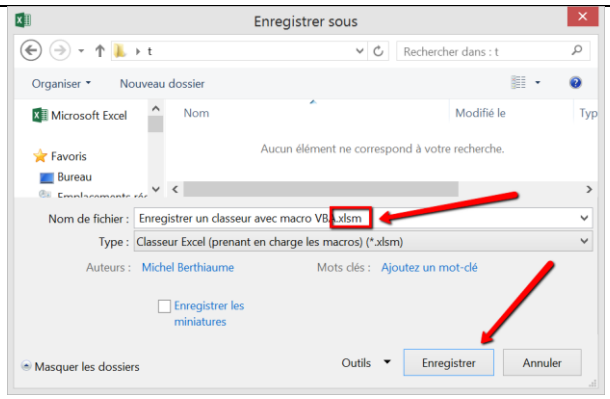
sera affiché.

3 Il suffit alors de cliquer sur Activer le contenu pour activer l'environnement VBA.

Enregistrer un classeur Excel contenant une macro VBA.docx

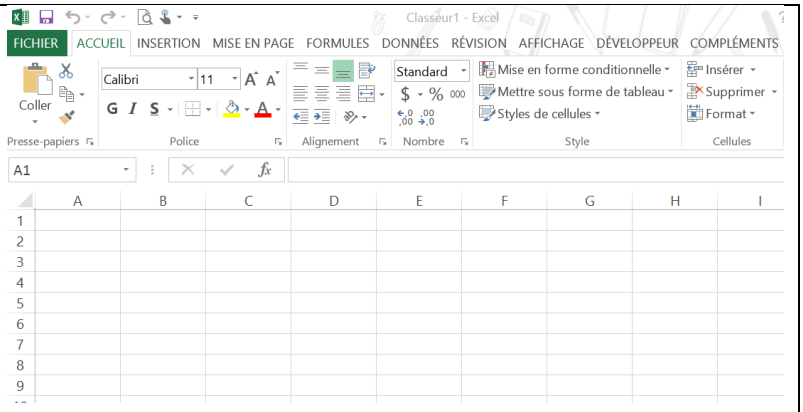
0	Dans un classeur Excel	
1	Contenant une macro VBA	
2	Enregistrer le classeur Excel	

4	Lire le message qui apparaît. ATTENTION : le message n'apparaît que dans les versions Excel 2007 et suivantes.	
5	ATTENTION : le bouton par défaut est Oui, alors qu'il faut choisir Non	
6	Choisir le bon type de fichier	 

7	Et enregistrer le classeur.	
Suggestion : passer directement de l'étape 2 à l'étape 7 en choisissant immédiatement le bon type de fichier		

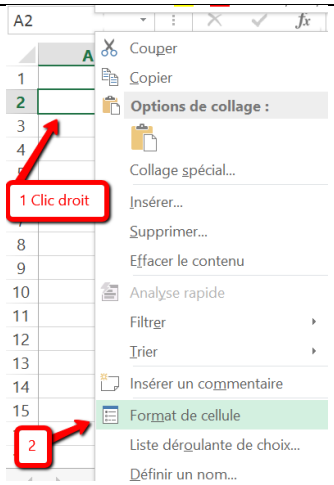
Déverrouiller des cellules Excel

0 Dans un nouveau classeur Excel



1 Cliquer une cellule A2 avec le bouton droit.

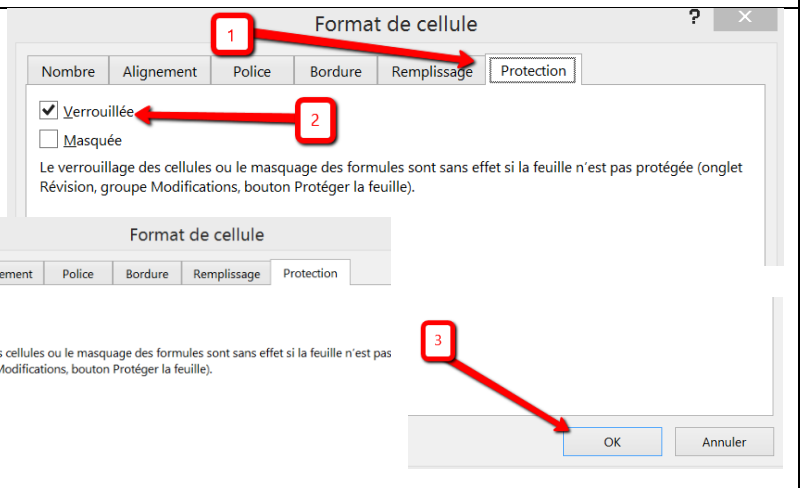
Cliquer Format de cellule.



2 Dans la boîte de dialogue Format de cellule :

Choisir l'onglet Protection.

Désactiver la case à cocher Verrouillée.



Cliquer Ok

3 Répéter les étapes 1 et 2 pour quelques cellules, B1 et D1 par exemple.

Modifier l'interface VBE

Il est fortement recommandé de faire les modifications suivantes à L'interface VBE :

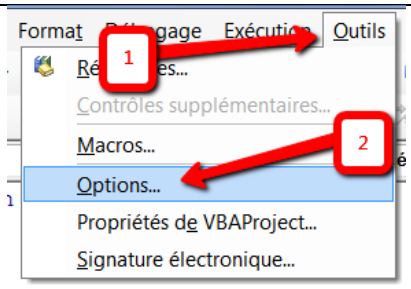
1 Rendre automatique Option Explicit :

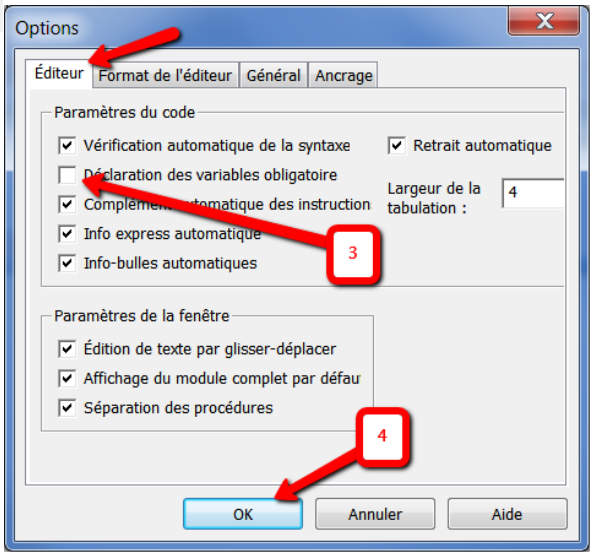
Cliquer Outils

Cliquer Options

Cocher

Cliquer Ok





2

Les boutons Commenter bloc  et

Ne pas commenter bloc  vous seront utiles pour rendre inopérantes/
opérantes des instructions erronées ou suspectes lors de tests et débogages.

Ils font partie de la barre de boutons débogage.
Vous pouvez aussi les ajouter à la barre d'accès rapide :

Enfoncer le bouton de droite de la souris sur une zone inutilisée de la barre d'accès rapide.

Choisir Personnaliser…

Ouvrir l'onglet **Commandes.**

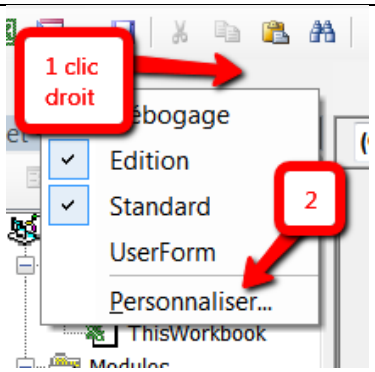
Choisir la Catégorie **Édition.**

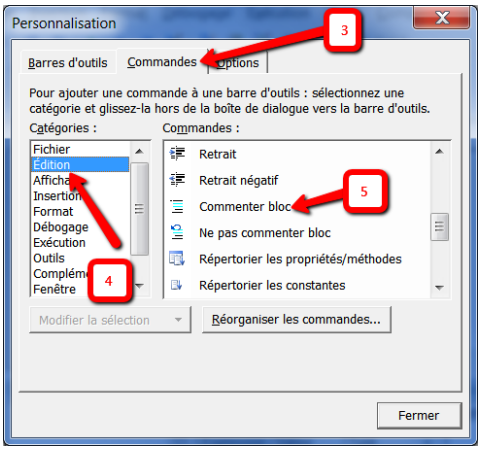
Dans la liste déroulante, choisir **Commenter bloc**.

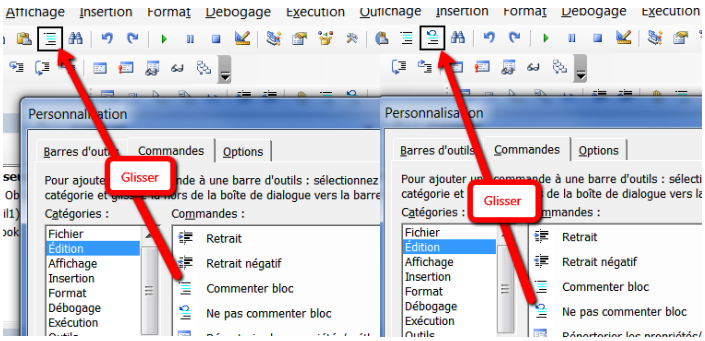
Glisser l'icône à l'endroit choisi de la barre d'accès rapide.

Répéter pour le bouton **Ne pas commenter bloc**.

Cliquer 







Installer l'aide VBA locale

Microsoft a publié l'aide VBA Excel de différentes façons selon les versions d'Excel (et d'office). Voici quelques fichiers intéressants:

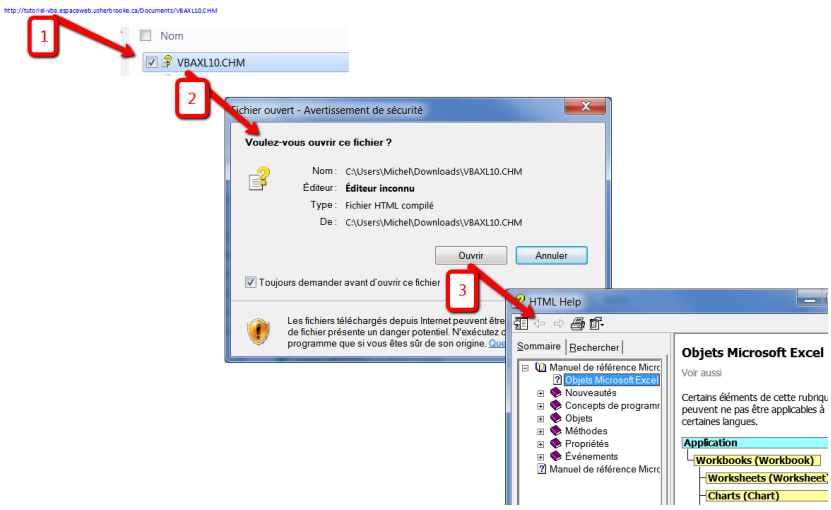
Office 2013 (en anglais):

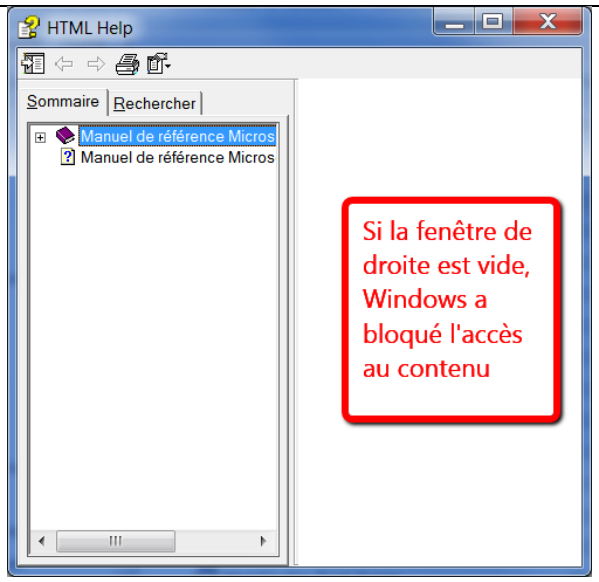
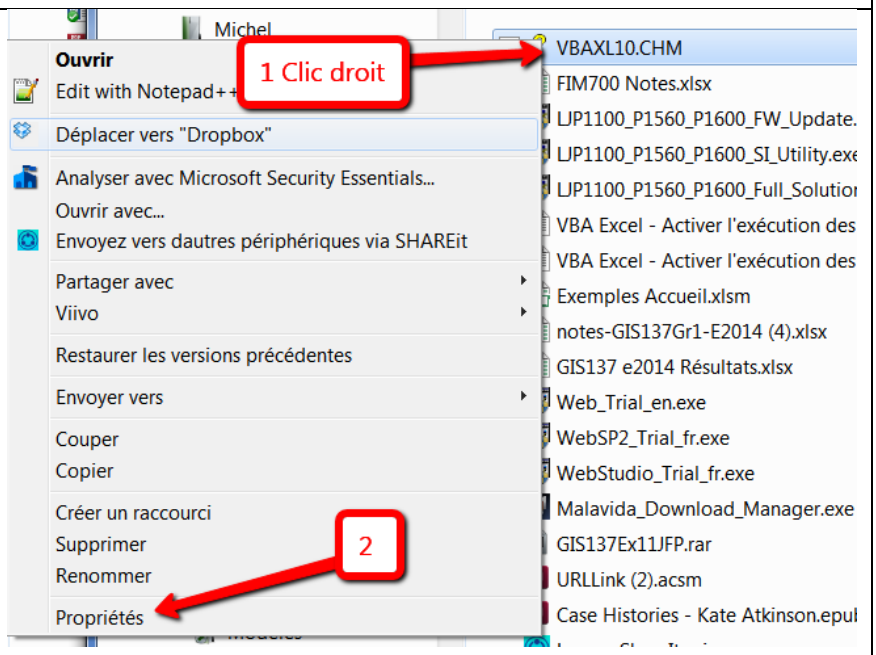
- Office Shared 2013 Developer Documentation
- Excel 2013 Developer Documentation

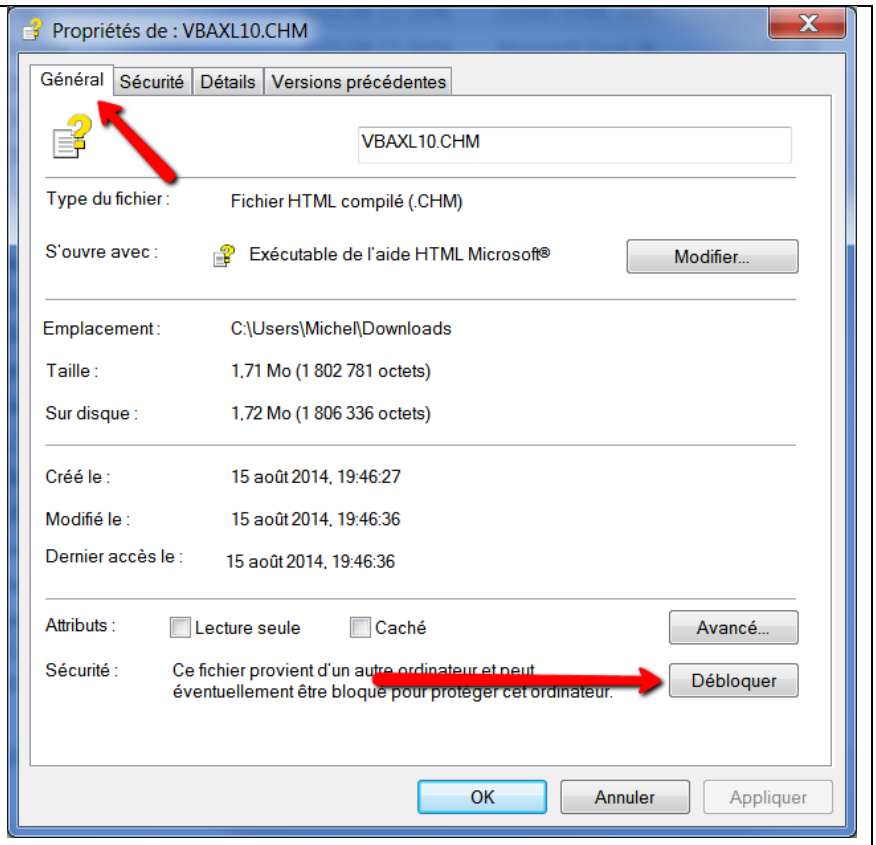
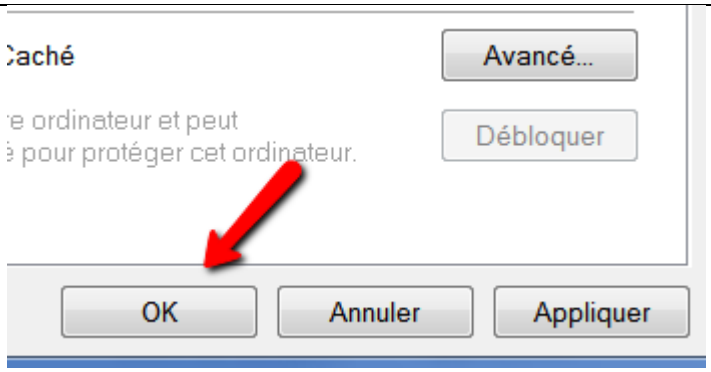
Office 2003 (en français):

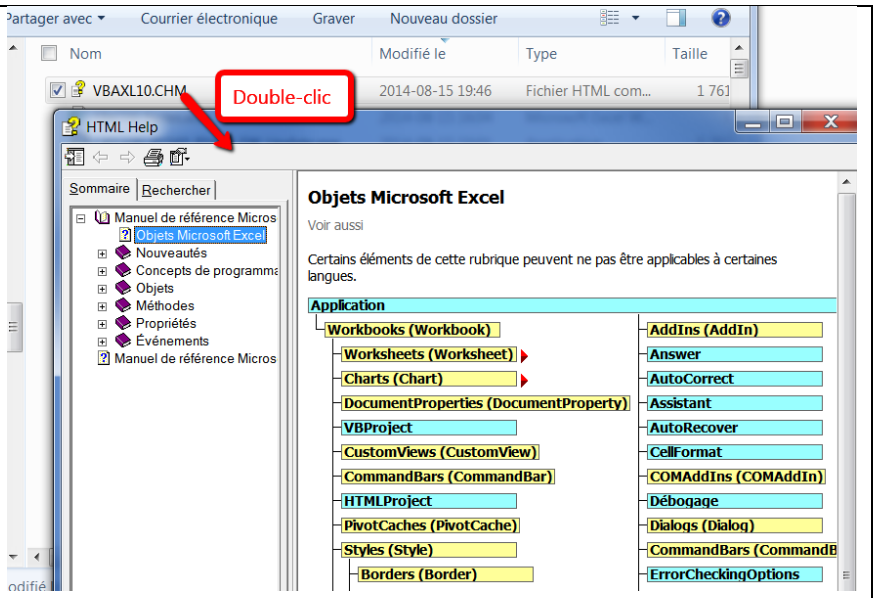
- Référence du langage VBA
- Référence de l'interface de développement (VBE)
- Modèle objet Microsoft Excel

Noter qu'il y a très peu de différences entre VBA pour Office 2003 et VBA pour Office 2013.

1	Télécharger et ouvrez le fichier choisi en cliquant sur la référence ci-dessus Enregistrer le fichier dans un dossier de votre ordinateur que vous retrouverez facilement Ouvrir le fichier.	Exemple : Modèle objet Microsoft Excel 

	Le fichier est peut-être bloqué parce qu'il provient d'un autre ordinateur :	
2	Pour le débloquer : Afficher les propriétés du fichier :	

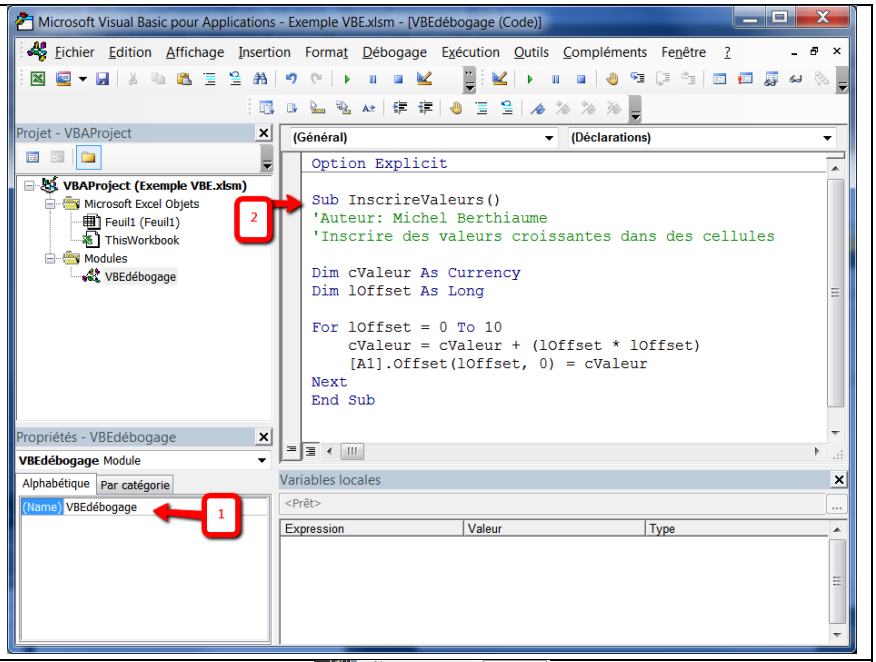
3	Dans l'onglet Général, cliquer le bouton Débloquer	
4	Cliquer Ok	

5	Ouvrir le fichier d'aide	

Déboguer un programme VBA :

0 Copier-coller le code ci-dessous dans un module VBA Excel :

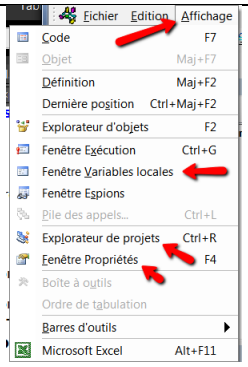
```
Sub InscrireValeurs()
'Auteur: Michel Berthiaume
'Inscrire des valeurs croissantes dans des cellules

Dim cValeur As Currency
Dim lOffset As Long

For lOffset = 0 To 10
   cValeur = cValeur + (lOffset * lOffset)
   [A1].Offset(lOffset, 0) = cValeur
Next
End Sub
```



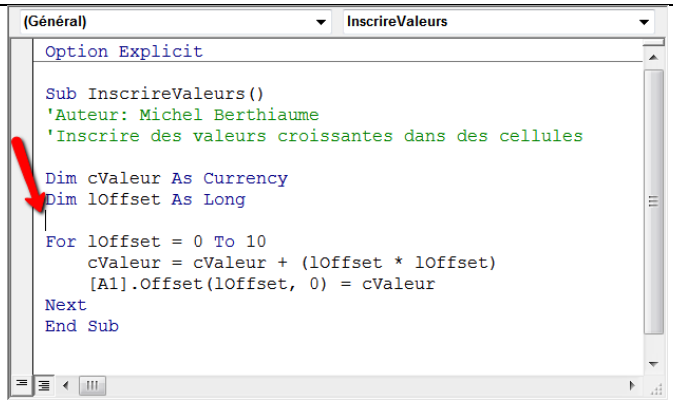
S'assurer que les fenêtres

- Projet
- Propriétés
- Code
- Variables locales

sont affichées



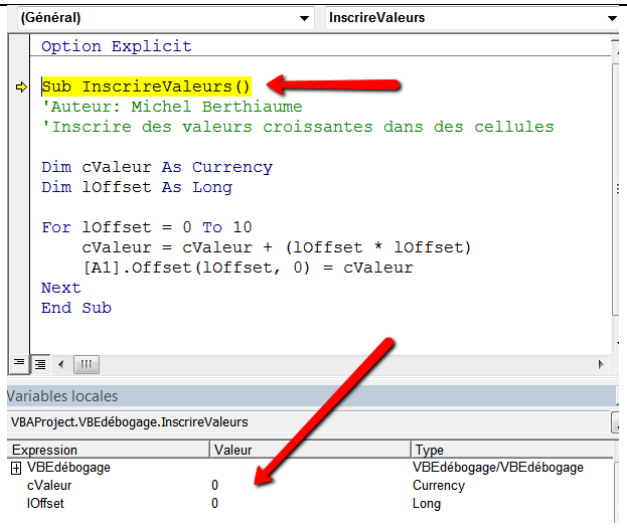
1 Placer le curseur dans la procédure.



3

Appuyer la touche F8 pour exécuter une ligne de code.

Examiner les résultats de cette exécution dans la fenêtre variables.



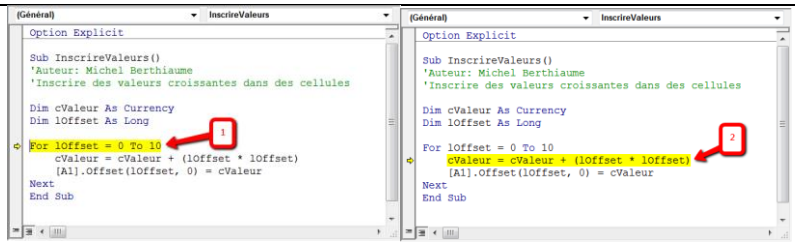
4

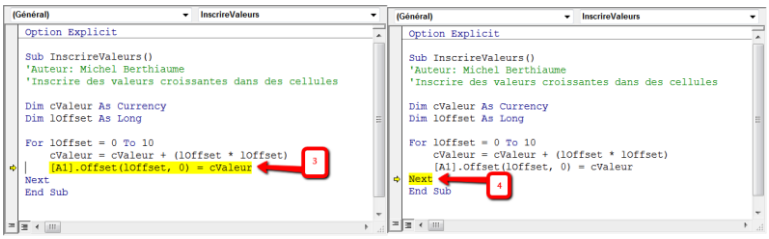
Appuyer la touche F8 à nouveau 5 fois :

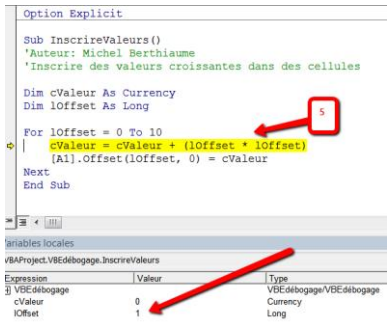
La barre d'exécution jaune se déplace d'une ligne à l'autre.

Remarquer que les instructions à l'intérieur des balises For/Next sont répétées.

Remarquer que les valeurs de cValeur et lOffset changent.





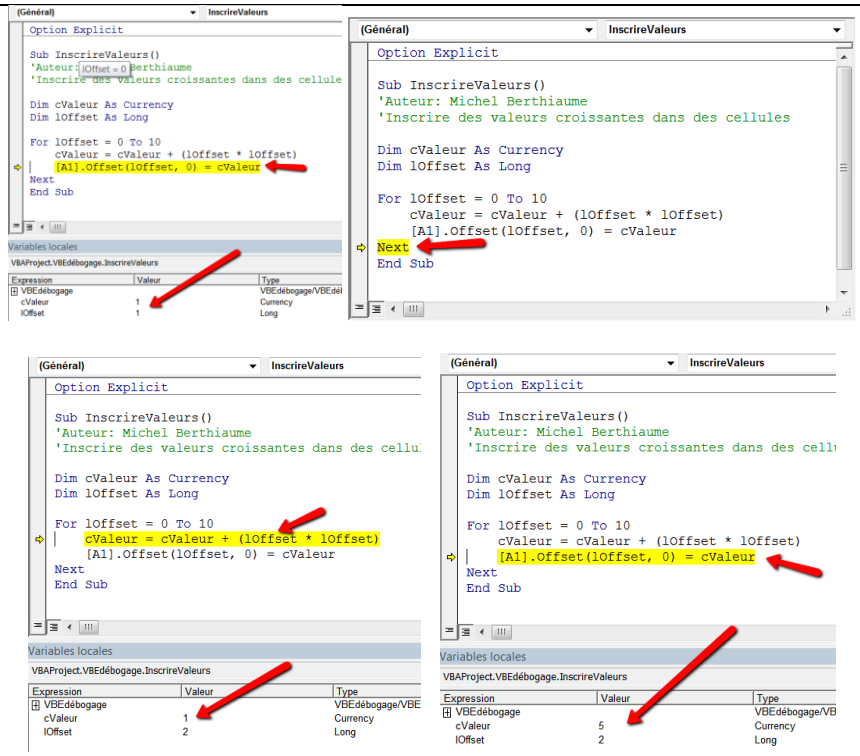


5

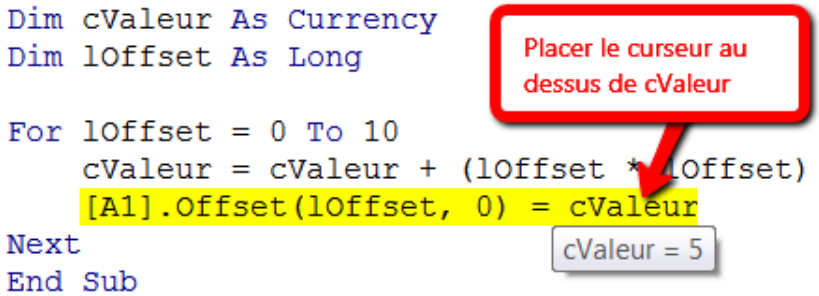
Appuyer la touche

à nouveau 4 fois :

Remarquer que les valeurs des variables changent.



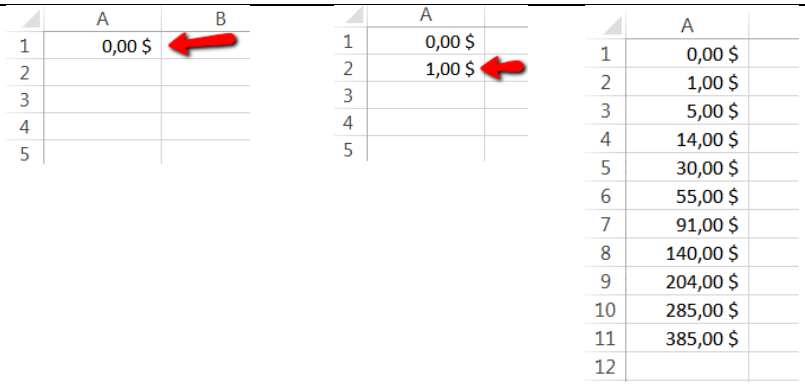
On peut connaître la valeur d'une variable en la survolant avec le curseur.



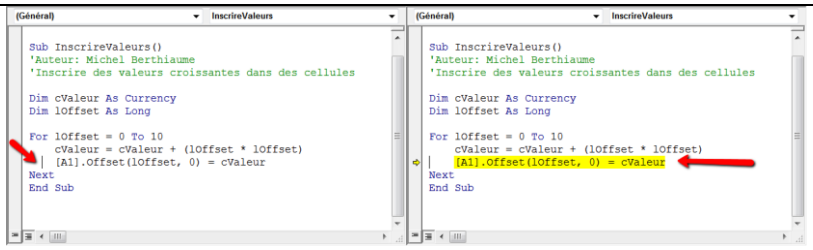
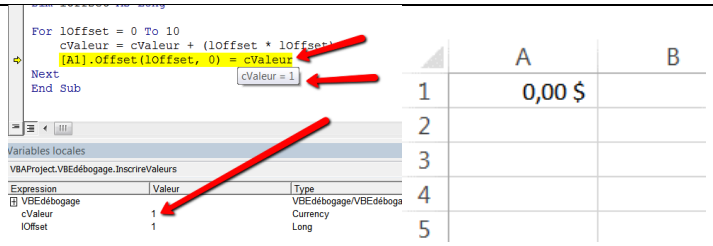
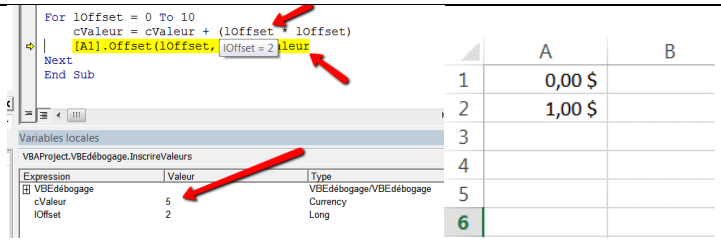
6

Remarquer que des valeurs s'inscrivent dans le classeur Excel :

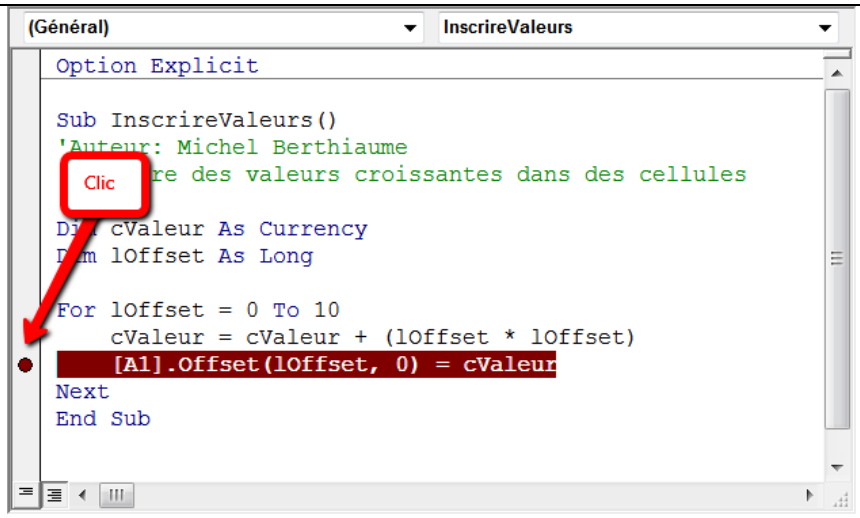
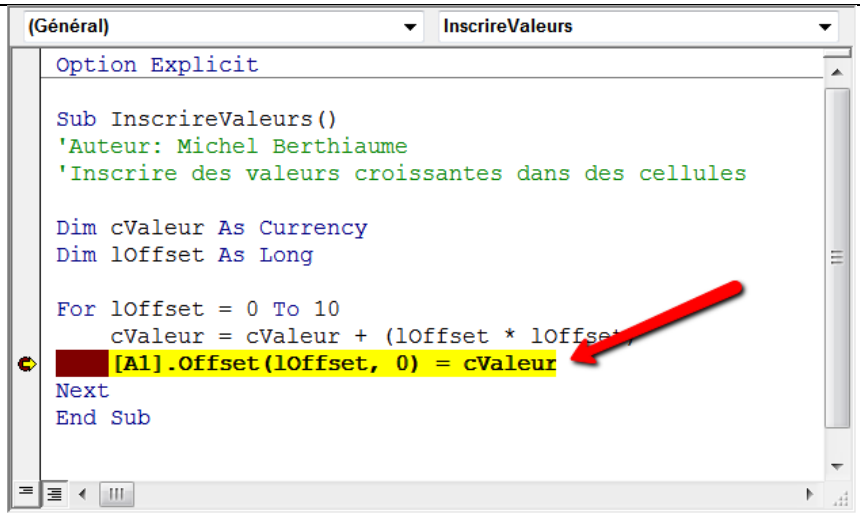
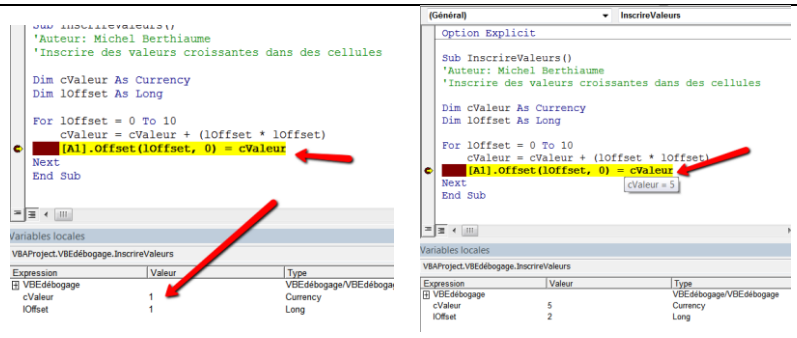
Répéter pour l'évolution du contenu des variables et du classeur Excel.



ALTERNATIVE :

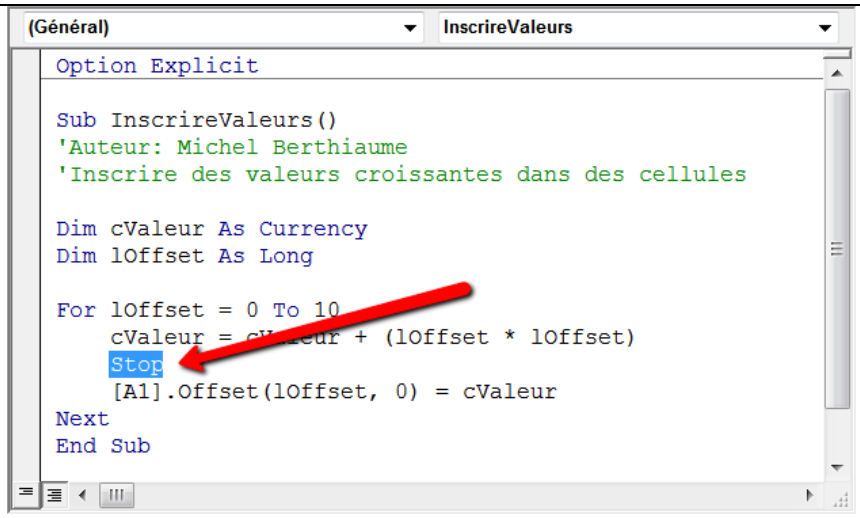
1	Placer le curseur dans la procédure, sur la ligne qui vous inquiète. Appuyer les touches Ctrl + F8 pour exécuter la procédure jusqu'à cette ligne.	
2	Appuyer à nouveau les touches Ctrl + F8 Examiner les résultats de cette exécution dans la fenêtre variables locales et dans la fenêtre espions, en survolant une variable de la procédure avec la souris ou dans le classeur Excel.	
3	Appuyer à nouveau les touches Ctrl + F8 Examiner les résultats de cette exécution dans la fenêtre variables locales et dans la fenêtre espions, en survolant une variable de la procédure avec la souris ou dans le classeur Excel.	

ALTERNATIVE :

1	Placer un point d'arrêt dans la marge à la hauteur de la ligne qui vous inquiète en cliquant dans la marge On peut aussi placer le curseur dans la ligne et enfoncer F9 ou Menu Débogage/point d'arrêt.	
2	Appuyer la touche F5 pour exécuter la procédure jusqu'à cette ligne.	
3	Appuyer à nouveau la touche F5 pour poursuivre l'exécution de la procédure jusqu'à cette ligne. Il est aussi possible d'utiliser la touche F8 pour exécuter le programme une ligne à la fois.	

ALTERNATIVE :

1 Placer l'instruction **Stop** dans une nouvelle ligne avant la ligne qui vous inquiète.



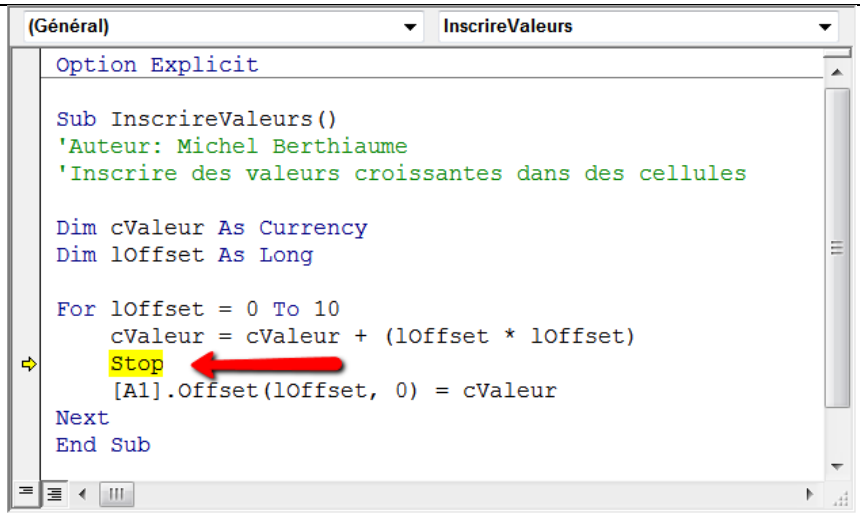
2 Démarrer l'exécution avec F5.

Elle s'arrête à la line **Stop**.

Appuyer la touche F8 pour exécuter la ligne de code suivante.

Répéter F8 pour chaque ligne que vous voulez exécuter.

Ou encore utiliser F5 pour relancer l'exécution jusqu'à l'instruction Stop.



Si l'exécution d'un programme VBA semble s'éterniser:

- Stopper l'exécution avec les touches 

+ 

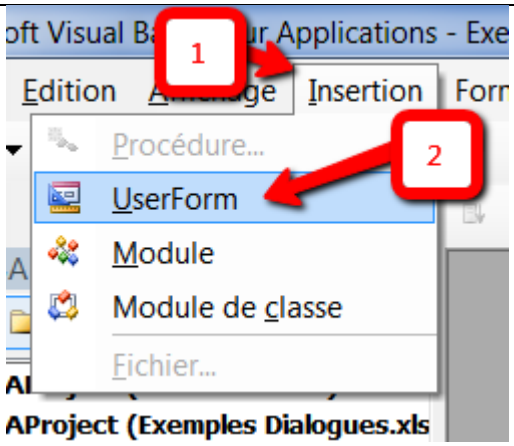
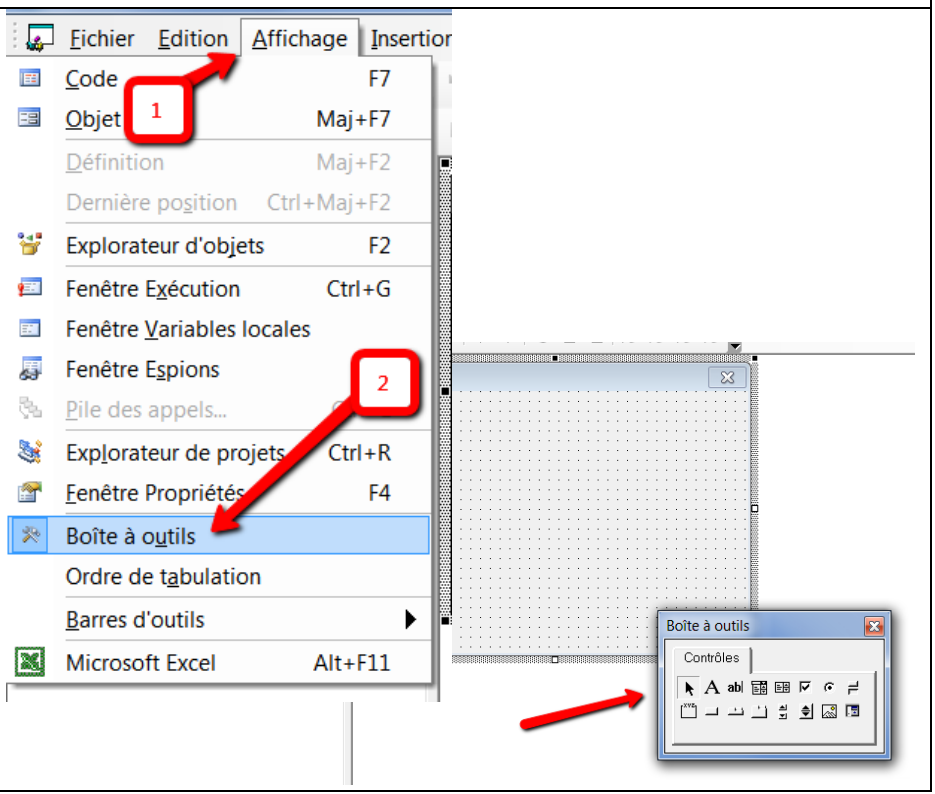
.
- Utiliser la touche 

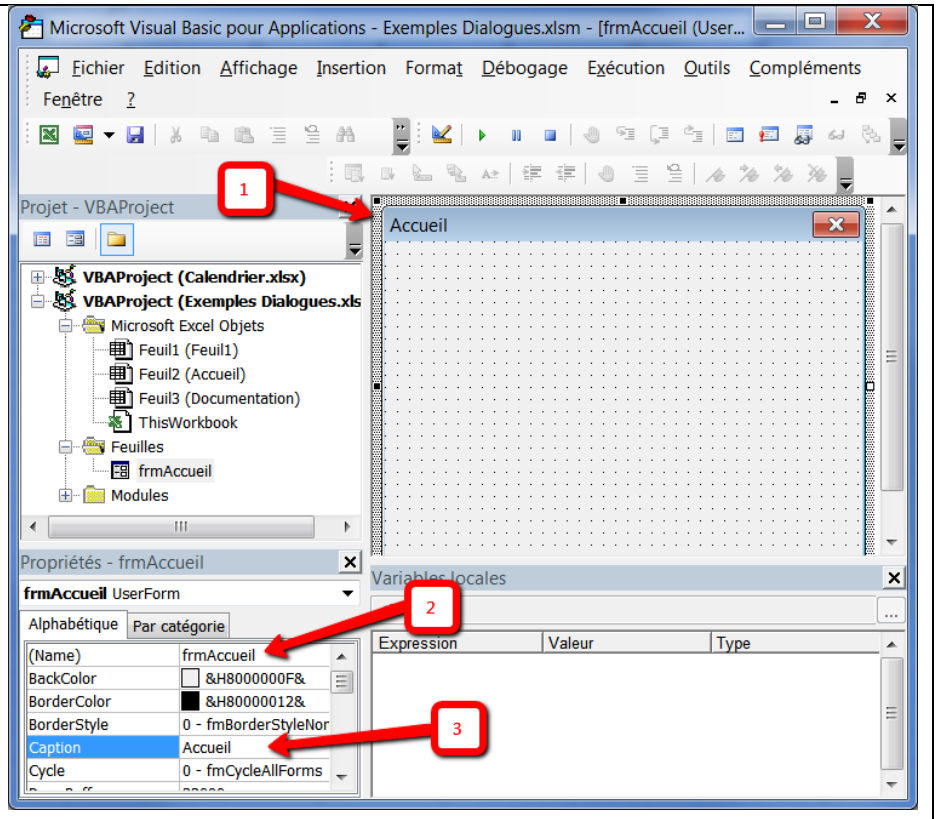
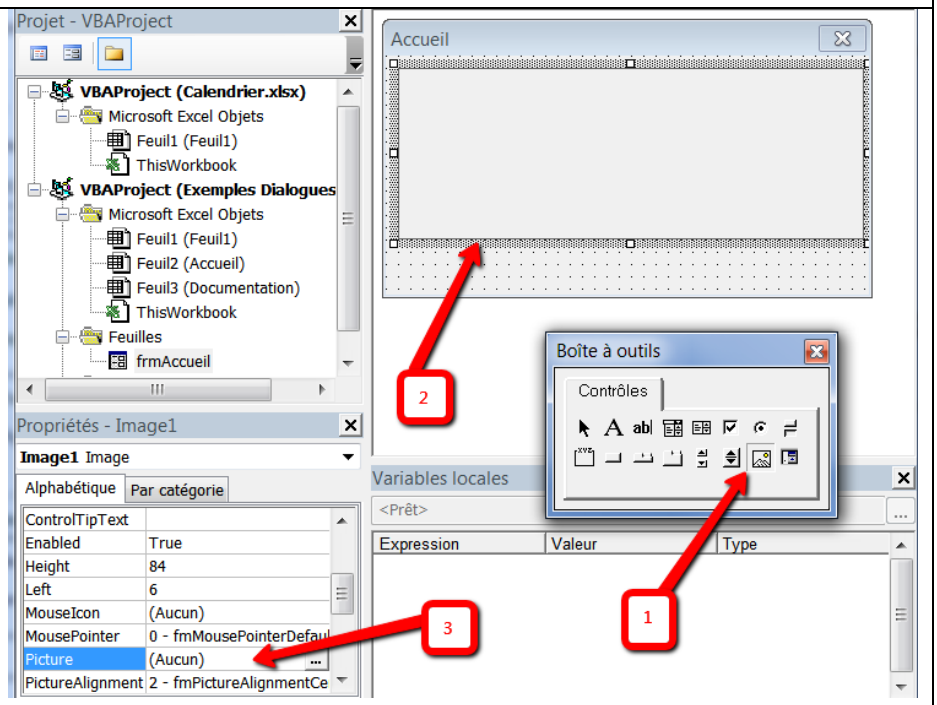
pour exécuter les lignes de code une à une.
- Examiner les résultats de l'exécution de chaque ligne dans la fenêtre variables locales et dans la fenêtre espions, en survolant une variable de la procédure avec la souris ou dans le classeur Excel.

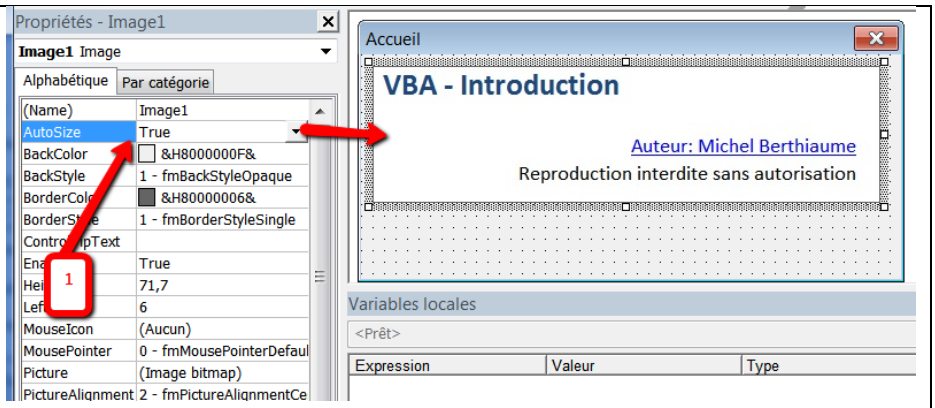
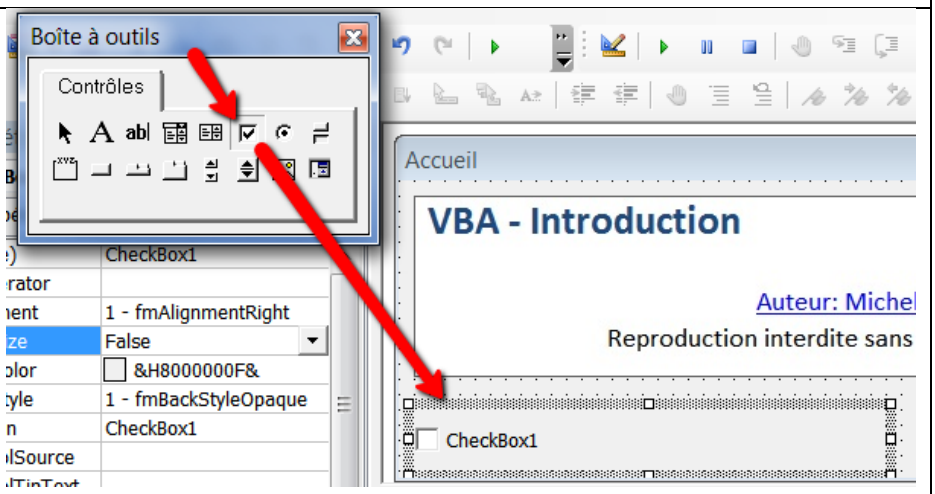
Pour une procédure Sub avec paramètre ou pour une procédure Function (fonction):

- Placer un point d'arrêt près d'une ligne suspecte de la procédure ou la fonction.
  OU
  Placer l'instruction **Stop** près d'une ligne suspecte de la procédure ou la fonction.
- Exécuter la procédure ou la fonction en l'appelant à partir de la fenêtre Exécution.

Créer un formulaire (UserForm) d'accueil

1	Dans VBE : 1. Menu insertion 2. Choix UserForm	
2	Afficher la boîte à outils si elle n'apparaît pas déjà: 1. Menu Affichage 2. Boîte à outils	

3	Modifier les principales propriétés du UserForm : 1. Sélectionner le formulaire 2. Changer la propriété Name 3. Changer la propriété Caption	
4	Ajouter une image 1. Sélectionner le contrôle Image 2. Dessiner la forme dans le formulaire 3. Importer le fichier image	

4	Ajuster l'image à la forme	
5	Ajouter la case à cocher	
6	Modifier la case à cocher 1- Son nom (Name) 2- Son libellé (Caption) 3- La cellule Excel correspondante (ControlSource)