Le Tableur EXCEL
La Programmation en VBA

U.F.R. d'informatique

Juliette Dibie

PLAN

I. INTRODUCTION A EXCEL VBA

EXCEL VBA (Visual Basic pour **Application)** est un langage de programmation permettant d'utiliser du code **V**isual **B**asic pour exécuter les nombreuses fonctionnalités de l'**A**pplication EXCEL.

Un programme écrit en VBA est souvent appelé une **macro**.

Les macros permettent notamment d'automatiser des tâches répétitives réalisées sous EXCEL. Elles peuvent aussi être utilisées pour créer des boîtes de dialogue afin de rendre une application développée sous EXCEL plus conviviale.

Une macro peut être créée en utilisant l'**enregistreur de macros**, qui ne nécessite aucune connaissance du langage VBA.

Cependant une macro ainsi créée ne s'exécutera que sur un ensemble de cellules données et le code produit ne sera pas toujours très efficace. Pour pouvoir créer des macros propres à ses besoins, efficaces et interactives, il faut apprendre à programmer en VBA.

II. L'ENREGISTREUR DE MACRO

L'enregistreur de macro permet d'écrire du code VBA à partir d'opérations effectuées manuellement sous EXCEL.

II.1. ENREGISTRER UNE MACRO

Créer une macro MACRO1 qui écrit dans la cellule A1 le texte "AgroParisTech" en caractères gras et dans la cellule G1 la date du jour en caractères italiques.

Avant de commencer l'enregistrement, il faut se poser plusieurs questions :

- penser à la manière d'effectuer les opérations manuelles ;
- se demander quand commencer l'enregistrement. Dans notre cas, la sélection de la cellule A1 fait partie de l'enregistrement puisqu'on veut que le texte "AgroParisTech" se trouve dans cette cellule. Si l'enregistrement ne commence pas par la sélection de la cellule A1, le texte tapé est écrit dans la cellule active.
- se demander quand arrêter l'enregistrement.

  Question : que faire pour que la cellule active après l'exécution de la macro soit la cellule A1 ?

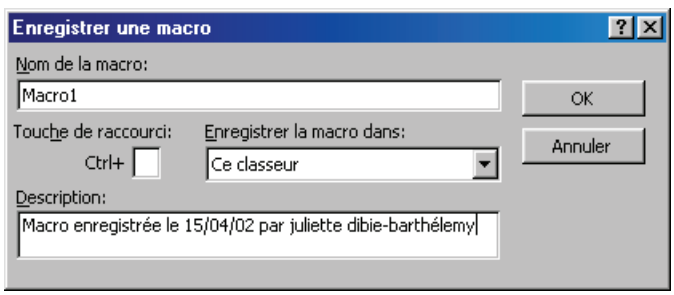
1) Ouvrir un nouveau classeur et l'enregistrer sous TEST-MACRO.XLS.

2) Les cellules dans une macro sont identifiées à l'aide d'une lettre (colonne) suivie d'un chiffre (ligne), comme par exemple la cellule A1. Afin de faciliter la lecture du code VBA généré, il est préférable de choisir la même identification des cellules dans le classeur dans lequel est enregistré la macro. Si tel n'est pas le cas, activer la commande OUTILS OPTIONS, cliquer sur l'onglet GENERAL et décocher la case STYLE DE REFERENCE L1C1.

3) Positionner le curseur sur une autre cellule que la cellule A1 de la feuille de calcul FEUIL1.

4) Activer la commande OUTILS MACRO NOUVELLE MACRO.

5) Taper MACRO1 dans la zone NOM DE LA MACRO.



6) Dans la zone ENREGISTRER LA MACRO DANS, sélectionner l'option CE CLASSEUR ; la macro MACRO1 n'est alors disponible que dans le classeur TEST-MACRO.XLS.

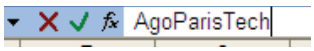
Remarque : La zone ENREGISTRER LA MACRO DANS permet de déterminer l'endroit où la macro MACRO1 sera enregistrée. Il existe trois options :

- Avec l'option CLASSEUR DE MACROS PERSONNELLES, la macro est enregistrée dans un classeur spécial appelé PERSO.XLS. Ce classeur est ouvert automatiquement lors du lancement d'EXCEL et ses macros sont donc disponibles dans tous les classeurs ouverts. La commande FENETRE AFFICHER permet d'afficher ce classeur et la commande FENETRE MASQUER de le masquer.
- L'option NOUVEAU CLASSEUR permet d'enregistrer la macro dans un nouveau classeur.
- L'option CE CLASSEUR permet d'enregistrer la macro dans le classeur actif.

7) Cliquer sur le bouton OK.

8) La barre d'outils ARRET DE L'ENREGISTREMENT apparaît, ce qui marque le début de l'enregistrement. Par défaut, l'enregistrement d'une macro s'effectue en utilisant des **références absolues aux cellules**.

Remarque : Il est possible d'enregistrer une macro en utilisant des références relatives, en cliquant sur le bouton REFERENCE RELATIVE de la barre d'outils ARRET DE L'ENREGISTREMENT.

9) Positionner le curseur sur la cellule A1, taper le texte "AgroParisTech", valider avec le bouton √ 

et sélectionner le style gras (**G**).

10) Positionner le curseur sur la cellule G1, taper la formule =AUJOURDHUI(), valider avec le bouton √ et sélectionner le style italique (***I***).

11) Arrêter l'enregistrement en cliquant sur le carré bleu de la barre d'outils ARRET DE L'ENREGISTREMENT.

NE JAMAIS OUBLIER D'ARRETER L'ENREGISTREMENT D'UNE MACRO.

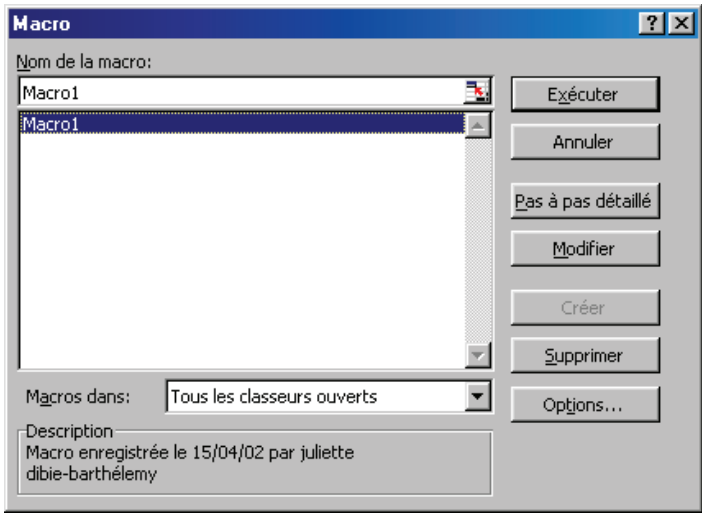
II.2. EXECUTER UNE MACRO

1) Effacer le contenu des cellules A1 et G1.

2) Positionner le curseur sur une autre cellule que la cellule A1 de la feuille de calcul FEUIL1.

3) Activer la commande OUTILS MACRO MACROS.

4) Sélectionner la macro MACRO1 et cliquer sur le bouton EXECUTER.



POUR SUPPRIMER UNE MACRO, ACTIVER LA COMMANDE OUTILS MACRO MACROS, SELECTIONNER LA MACRO A SUPPRIMER ET CLIQUER SUR LE BOUTON SUPPRIMER.

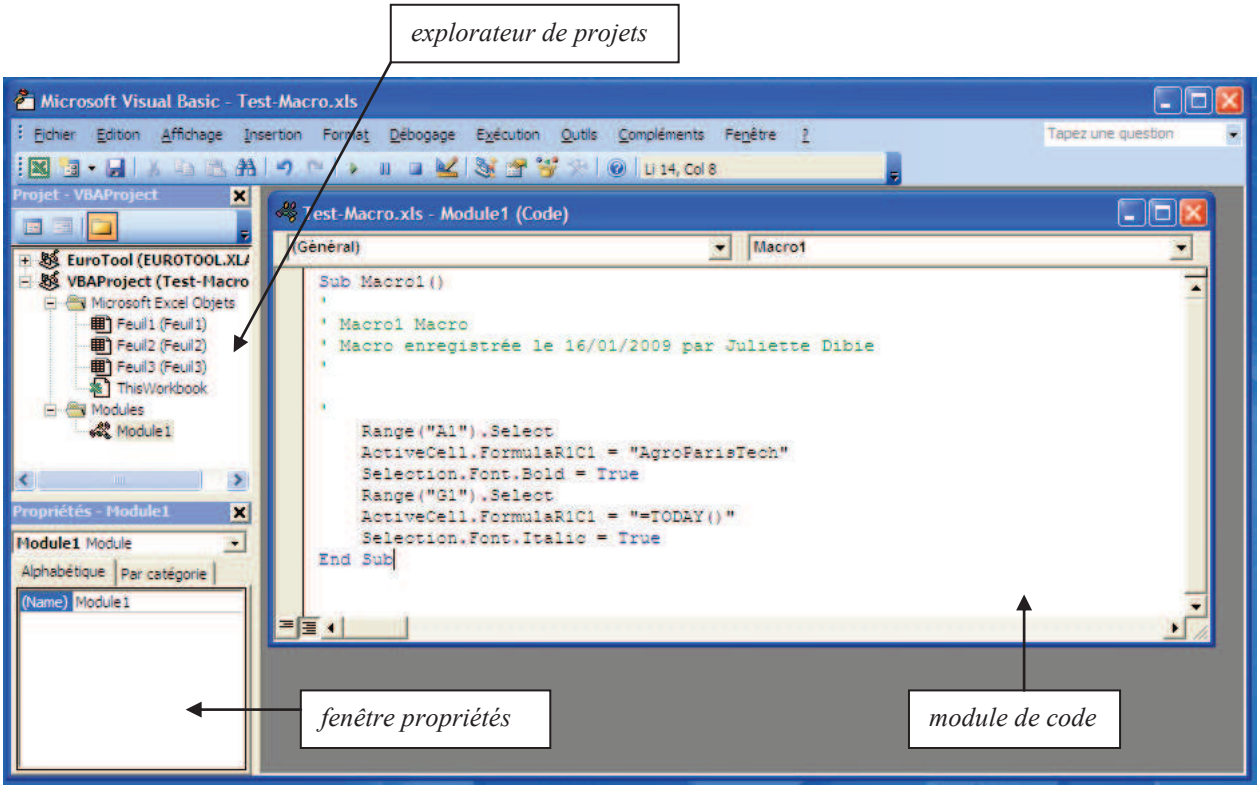
II.3. EXERCICE

Créer une macro MACROMOIS qui écrit les noms de mois Janvier, Février et Mars en colonne à partir de la cellule active. Utiliser l'enregistreur de macro avec référence relative aux cellules.

III. L'ENVIRONNEMENT VISUAL BASIC EDITOR

Nous avons vu comment créer une macro à l'aide de l'enregistreur de macro. Nous allons maintenant examiner le code VBA produit. Pour ce faire, il faut utiliser l'éditeur de Visual Basic, **VISUAL BASIC EDITOR (VBE)**, qui s'exécute dans une fenêtre différente de celle d'EXCEL.

Ouvrir VBE en activant la commande AFFICHAGE BARRE D'OUTILS VISUAL BASIC et en cliquant sur l'objet VISUAL BASIC EDITOR.



L'EXPLORATEUR DE PROJETS

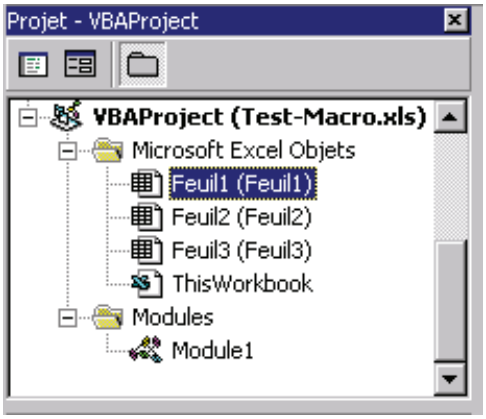
A chaque classeur EXCEL ouvert est associé un **projet VBA**.

L'**explorateur de projets** affiche une liste hiérarchisée des différents projets VBA associés aux classeurs EXCEL ouverts.

Un projet VBA associé à un classeur regroupe les éléments du classeur, comme ses feuilles de calcul ou des boîtes de dialogue, et les procédures et les fonctions associées au classeur et stockées dans un ou plusieurs modules de code.

Le projet VBA associé au classeur TEST-MACRO.XLS est composé de deux dossiers :

- le dossier MICROSOFT EXCEL OBJETS qui contient les éléments attachés au projet : le classeur TEST-MACRO.XLS (THISWORKBOOK) et ses feuilles de calcul FEUIL1, FEUIL2 et FEUIL3 ;
- le dossier MODULES qui contient les modules de code du projet : le module MODULE1 qui contient la macro MACRO1.



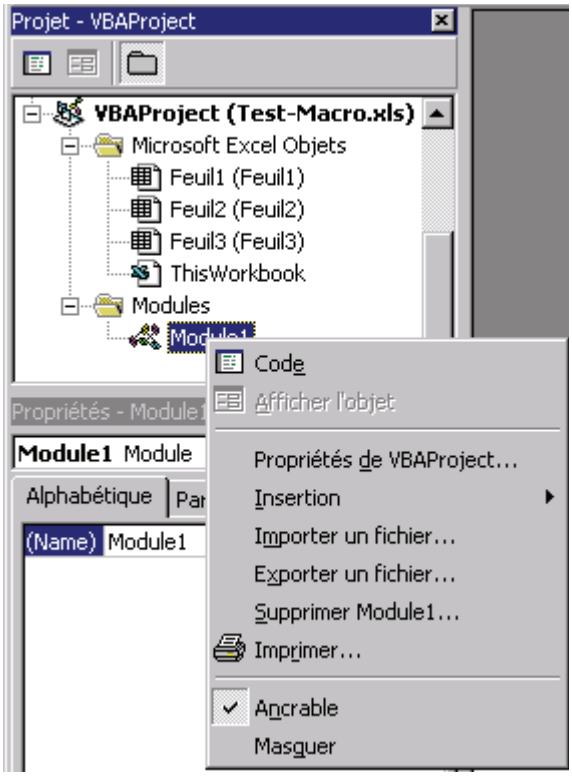
SAUVEGARDER REGULIEREMENT SON TRAVAIL A L'AIDE DE LA COMMANDE FICHIER ENREGISTRER "NOMCLASSEUR.XLS".

LA COMMANDE FICHIER FERMER ET RETOURNER A MICROSOFT EXCEL PERMET DE FERMER VBE ET DE RETOURNER DANS EXCEL.

III.2. LES MODULES DE CODE

L'explorateur de projets permet, à l'aide du clic droit de la souris, d'ouvrir un module de code (option CODE), d'insérer un nouveau module de code (option INSERTION) ou d'en supprimer un (option SUPPRIMER <NOMMODULE>).

La commande INSERTION MODULE permet également d'insérer un nouveau module de code.



Toutes les macros sont enregistrées dans un module de code. L'enregistreur de macro a inséré le module de code MODULE1 qui contient la macro MACRO1. Un module de code peut contenir plusieurs macros. On peut insérer autant de modules de code qu'on le désire.

III.3. LES PROCEDURES

Une macro est appelée en VBA une **procédure**, c'est une suite d'instructions qui ne retourne pas de valeur.
VBA permet également d'écrire des **fonctions**. Une fonction est une suite d'instructions qui retourne une valeur. Nous les étudierons plus loin.

L'enregistreur de macro ne génère que des procédures. Une procédure commence par le mot clé Sub suivi du nom de la procédure et d'une liste d'arguments entre parenthèses, qui peut être vide. Elle se termine par le mot clé End Sub. Une procédure a la syntaxe suivante :

```
Sub NomProcédure([argument_1,..., argument_n])
   Instructions
   ...
End Sub
```

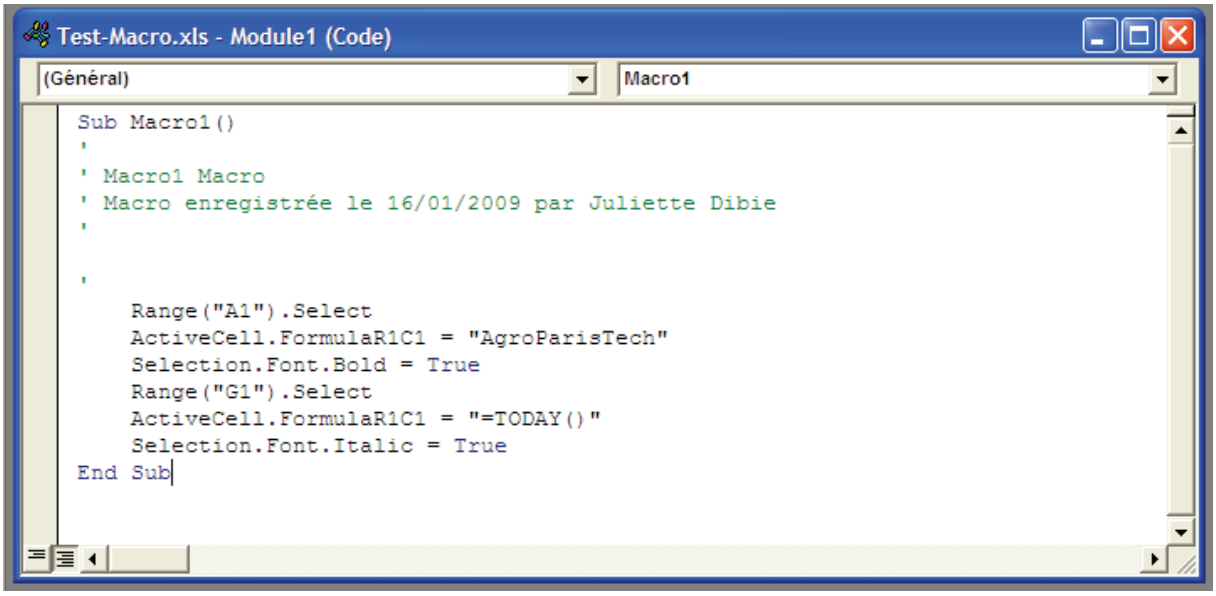
Remarque : Les crochets **[ ]** signifient que les arguments sont facultatifs.

Une **instruction** exécute une tâche précise. Elle est généralement écrite sur une seule ligne.
Exemple :

```
Range("A1").Select
```

III.4. MODIFIER UNE MACRO

Modifier la macro MACRO1 pour que la date du jour soit écrite dans la cellule G2 en caractères gras et italiques.



Avant de modifier la macro, voici quelques indications sur ses instructions :

- Les lignes précédées d'une apostrophe ' et de couleur verte sont des commentaires.
- Les mots de couleur bleue sont des mots clés (Sub, End Sub, True).
- L'instruction Range("A1").Select sélectionne (Select) une cellule (Range) désignée par sa ligne (1) et sa colonne (A)
- L'instruction ActiveCell.FormulaR1C1 affecte une formule à la cellule active (ActiveCell).
- L'instruction Selection.Font permet d'appliquer un format de police (Font.Bold ou Font.Italic) à la cellule sélectionnée (Selection).

1) Pour que la date du jour soit écrite dans la cellule G2 (et non dans la cellule G1) en caractères gras et italiques (et non plus seulement en italique), il faut : (a) modifier l'instruction Range("G1").Select en remplaçant G1 par G2 ; (b) ajouter à la fin de la procédure l'instruction Selection.Font.Bold=True.
2) Activer la commande FICHIER ENREGISTRER TEST-MACRO.XLS.

3) Retourner dans EXCEL, effacer le contenu des cellules A1 et G1, puis exécuter la macro MACRO1 à l'aide de la commande OUTILS MACRO MACROS.

III.5. EXERCICE

La macro MACROMOIS doit avoir à peu près le code suivant :

```
Sub MacroMois()
   ActiveCell.FormulaR1C1 = "Janvier"
   ActiveCell.Offset(1, 0).Range("A1").Select
   ActiveCell.FormulaR1C1 = "Février"
   ActiveCell.Offset(1, 0).Range("A1").Select
   ActiveCell.FormulaR1C1 = "Mars"
End Sub
```

L'instruction ActiveCell.Offset(1,0).Range("A1").Select sélectionne (Select) une cellule (Range) qui est sur une ligne en dessous et sur la même colonne (Offset(déplacement_ligne, déplacement_colonne)) que la cellule active (ActiveCell).

L'instruction ActiveCell.Offset(1,0).Range("A1").Select peut être remplacée par l'instruction ActiveCell.Offset(1,0).Select.

Modifier la macro pour que le nom de mois Avril soit écrit, en caractères gras et italiques, dans la cellule qui se trouve dans la colonne de droite et sur la même ligne que la cellule où est écrit Janvier.

Janvier	***Avril***
Février	
Mars	

IV. LES DIVERS MOYENS POUR EXECUTER UNE MACRO

Nous vous présentons plusieurs moyens pour exécuter une macro dans EXCEL de manière plus conviviale qu'à partir de la commande OUTILS MACRO MACROS.

IV.1. EXECUTER UNE MACRO A PARTIR D'UN RACCOURCI CLAVIER

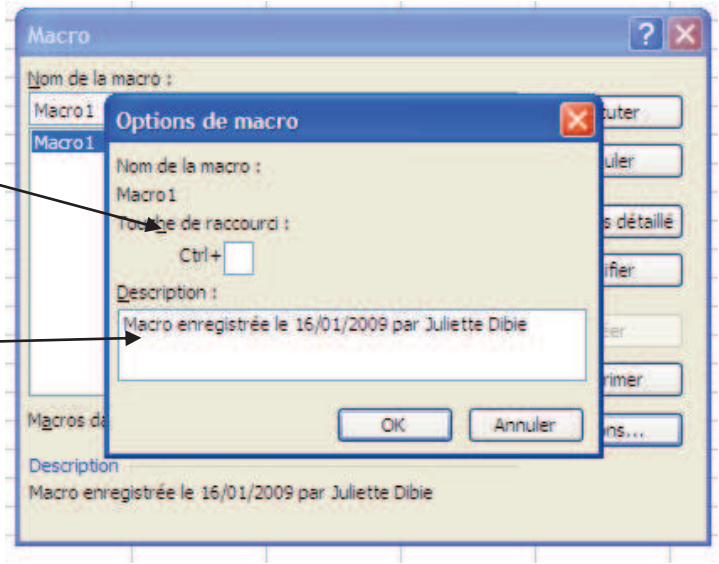
1) Retourner dans EXCEL et activer la commande OUTILS MACRO MACROS.

2) Sélectionner la macro MACRO1 et cliquer sur le bouton OPTIONS…

3) Taper **g** dans la zone TOUCHE DE RACCOURCI.

4) La zone DESCRIPTION permet de saisir un texte explicatif.

5) Cliquer sur le bouton OK.



6) Fermer la fenêtre MACRO en cliquant sur son bouton ANNULER ou sur son bouton de fermeture (**5**).

7) Vérifier que le raccourci clavier **Ctrl+g** provoque l'exécution de la macro MACRO1.

SI LE RACCOURCI CLAVIER CHOISI ETAIT DEJA AFFECTE A UNE COMMANDE EXCEL, IL SERA REATTRIBUE A LA MACRO SANS QUE L'ON EN SOIT INFORME.

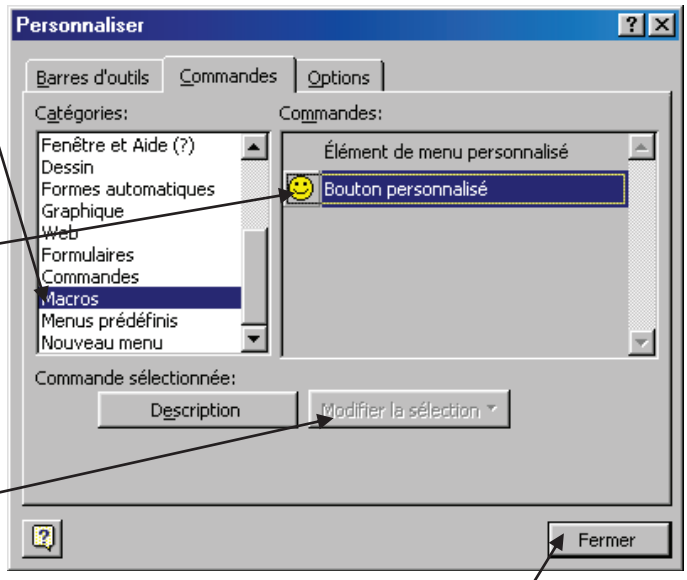
IV.2. EXECUTER UNE MACRO A PARTIR D'UN BOUTON D'UNE BARRE D'OUTILS

1) Activer la commande AFFICHAGE BARRE D'OUTILS PERSONNALISER…

2) Cliquer sur l'onglet COMMANDES et sélectionner l'option MACROS dans la liste CATEGORIES.

3) Sélectionner BOUTON PERSONNALISE dans la liste COMMANDES et le faire glisser vers la barre d'outils.

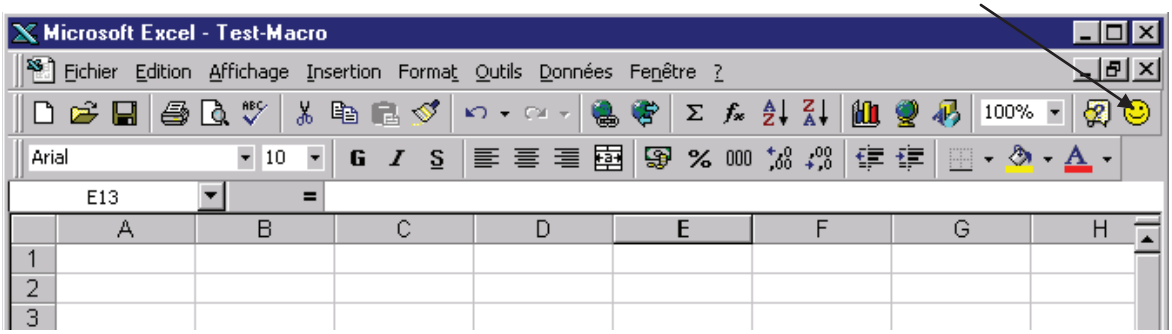
4) Le bouton MODIFIER LA SELECTION devient alors actif. Cliquer dessus et activer la commande AFFECTER UNE MACRO…



5) Sélectionner la macro MACRO1 et cliquer sur le bouton OK.

6) Cliquer sur le bouton FERMER de la fenêtre PERSONNALISER.

nouveau bouton dans la barre d'outils

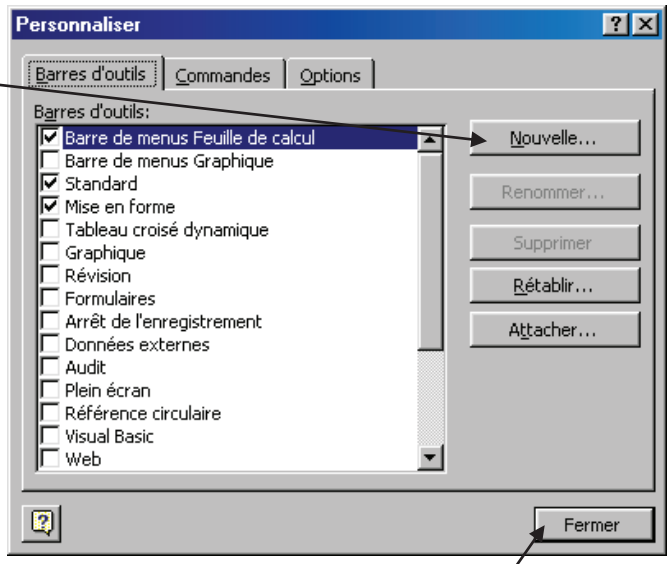


POUR SUPPRIMER UN BOUTON D'UNE BARRE D'OUTILS, ACTIVER LA COMMANDE AFFICHAGE BARRE D'OUTILS PERSONNALISER…, SELECTIONNER LE BOUTON DANS LA BARRE D'OUTILS, CLIQUER SUR LE BOUTON DROIT DE LA SOURIS ET ACTIVER LA COMMANDE SUPPRIMER. CLIQUER SUR LE BOUTON FERMER DE LA FENETRE PERSONNALISER.

COMMENT CREER UNE NOUVELLE BARRE D'OUTILS ?

1) Activer la commande AFFICHAGE BARRE D'OUTILS PERSONNALISER…

2) Cliquer sur l'onglet BARRES D'OUTILS et sur le bouton NOUVELLE…



3) Dans la zone NOM DE LA BARRE D'OUTILS taper MA BARRE et cliquer sur le bouton OK.

4) Une nouvelle barre d'outils vide apparaît.

5) On peut ajouter dans cette nouvelle barre d'outils un bouton pour exécuter la macro MACRO1.

6) Cliquer sur le bouton FERMER de la fenêtre PERSONNALISER.

POUR SUPPRIMER UNE BARRE D'OUTILS, ACTIVER LA COMMANDE AFFICHAGE BARRE D'OUTILS PERSONNALISER…, CLIQUER SUR L'ONGLET BARRES D'OUTILS SELECTIONNER MA BARRE DANS LA LISTE BARRE D'OUTILS ET CLIQUER SUR LE BOUTON SUPPRIMER. CLIQUER SUR LE BOUTON FERMER DE LA FENETRE PERSONNALISER.

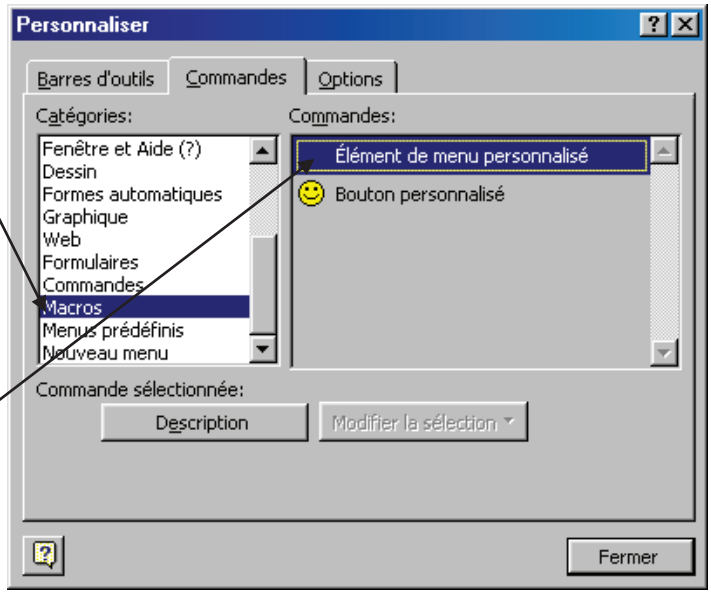
IV.3. EXECUTER UNE MACRO A PARTIR D'UN MENU

1) Activer la commande AFFICHAGE BARRE D'OUTILS PERSONNALISER…

2) Cliquer sur l'onglet COMMANDES et sélectionner l'option MACROS dans la liste CATEGORIES.

3) Dans la barre de menu d'EXCEL, cliquer sur le menu FORMAT pour y insérer le nouveau sous-menu.

4) Sélectionner ELEMENT DE MENU PERSONNALISE dans la liste COMMANDES et le faire glisser vers le menu FORMAT, sous le sous-menu STYLE…

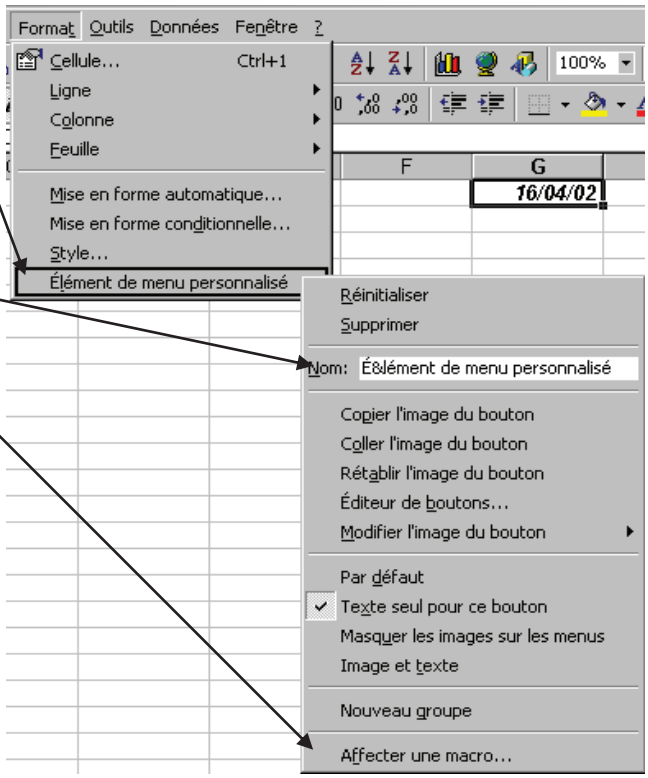


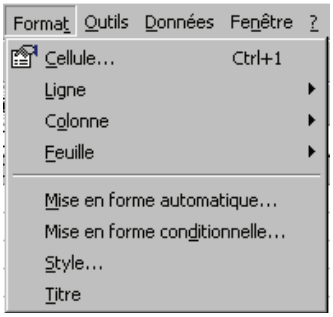
5) Sélectionner le nouveau sous-menu ELEMENT DE MENU PERSONNALISE et cliquer sur le bouton droit de la souris (affichage du menu contextuel).

6) Dans la zone NOM taper &TITRE (& provoque le soulignement du caractère qui suit).

7) Activer la commande AFFECTER UNE MACRO…, sélectionner la macro MACRO1 et cliquer sur le bouton OK.

8) Cliquer sur le bouton FERMER de la fenêtre PERSONNALISER.



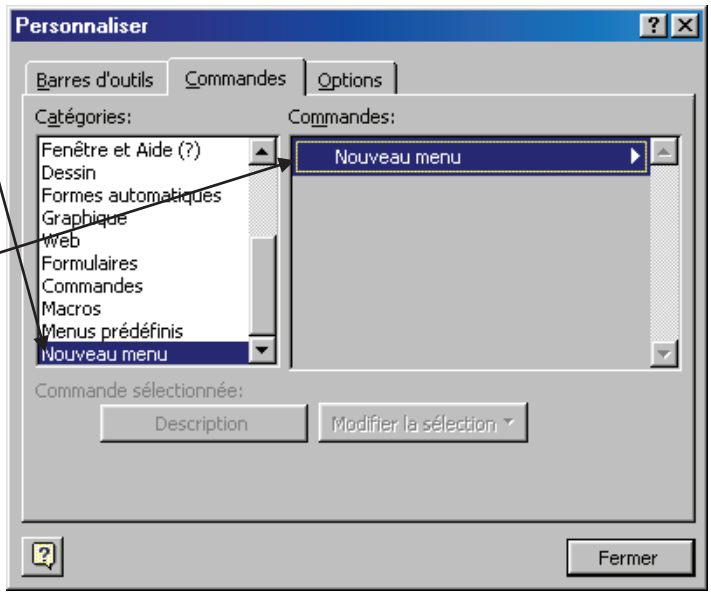


COMMENT CREER UN NOUVEAU MENU ?

1) Activer la commande AFFICHAGE BARRE D'OUTILS PERSONNALISER…

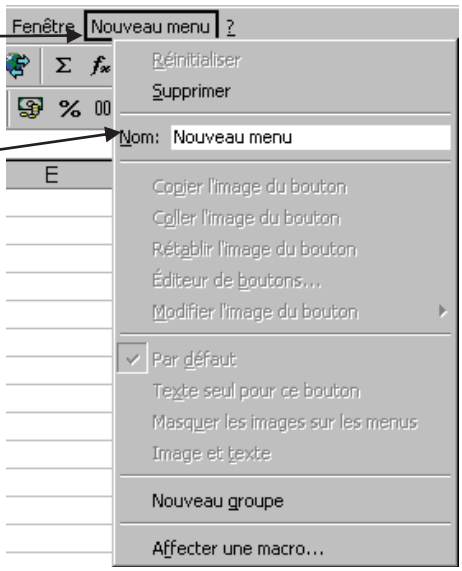
2) Cliquer sur l'onglet COMMANDES et sélectionner l'option NOUVEAU MENU dans la liste CATEGORIES.

3) Sélectionner NOUVEAU MENU dans la liste COMMANDES et le faire glisser vers la barre de menu entre les menus FENETRE et ? d'EXCEL.



4) Sélectionner le nouveau menu NOUVEAU MENU et cliquer sur le bouton droit de la souris.

5) Dans la zone NOM taper MENU &PERSO.



6) On peut ajouter dans ce nouveau menu un sous-menu TITRE pour exécuter la macro MACRO1.

7) Cliquer sur le bouton FERMER de la fenêtre PERSONNALISER.

POUR SUPPRIMER UN MENU (SOUS-MENU), ACTIVER LA COMMANDE AFFICHAGE BARRE D'OUTILS PERSONNALISER…, SELECTIONNER LE MENU (SOUS-MENU), CLIQUER SUR LE BOUTON DROIT DE LA SOURIS ET ACTIVER LA COMMANDE SUPPRIMER. CLIQUER SUR LE BOUTON FERMER DE LA FENETRE PERSONNALISER.

V. LE MODELE OBJET D'EXCEL

Le langage de programmation VBA dépend de l'application MICROSOFT OFFICE à laquelle il est attaché (WORD, EXCEL, ACCESS...).

Chaque application de MICROSOFT OFFICE possède un ensemble d'objets clairement définis, organisés en fonction des relations qui les unissent. Cette organisation s'appelle le **modèle objet** de l'application. Par exemple, dans WORD, les objets manipulés sont des documents, des paragraphes, des mots... Dans EXCEL, les objets manipulés sont des classeurs, des feuilles de calcul, des plages, des cellules...

Nous présentons, dans ce chapitre, le modèle objet d'EXCEL.

V.1. LES OBJETS

VBA est un **langage de programmation orientée objet** attaché à une application. Tout est objet.
Exemple d'objets :

- EXCEL est un objet Application ;
- un classeur est un objet Workbook ;
- une feuille de calcul est un objet Worksheet ;
- une plage de cellules (qui peut se limiter à une cellule) est un objet Range.

V.2. LES COLLECTIONS

De nombreux objets appartiennent à une **collection d'objets**, la collection étant elle-même un objet.
Exemple de collections :

- Dans l'objet Application, il existe une collection Workbooks qui contient tous les objets Workbook ouverts. Chaque objet Workbook comporte une collection Worksheets qui contient tous les objets Worksheet de ce classeur. Chaque objet Worksheet contient des objets Range.

V.3. L'ACCES AUX OBJETS

VBA permet de faire référence à un objet de différentes façons. Nous illustrons l'accès aux objets EXCEL à travers plusieurs exemples.

Pour faire référence à une plage de cellules donnée, il faut utiliser l'objet Range.
Exemple : Range("A1:B4") permet de faire référence à la plage de cellules A1:B4 et renvoie un objet Range.

Pour faire référence à un objet dans une collection, on peut soit utiliser le numéro de sa position dans la collection, soit son nom.
Exemple :
Worksheets(1) permet de faire référence à la première feuille de calcul du classeur actif et renvoie un objet Worksheet.
Workbooks("Classeur1.xls").Worksheets("Feuil1") permet de faire référence à la feuille de calcul de nom Feuil1 du classeur de nom Classeur1.xls et renvoie un objet Worksheet.

Remarque : La notation Workbooks("Classeur1.xls").Worksheets("Feuil1") est une notation raccourcie pour Application.Workbooks("Classeur1.xls"). Worksheets("Feuil1").
L'objet Application peut, en effet, toujours être omis.

V.4. EXERCICE

Ecrire l'instruction qui permet de rendre active la cellule B2 de la deuxième feuille de calcul de nom FEUIL2 du classeur TEST-MACRO.XLS.

V.5. LES PROPRIETES DES OBJETS

Chaque objet est défini par un ensemble de **propriétés**, qui représentent les caractéristiques de l'objet.

Pour faire référence à une propriété d'un objet donné, il faut utiliser la syntaxe Objet.Propriété.

Exemple :

- La propriété Value de l'objet Range désigne la valeur de l'objet spécifié. Range("A1").Value = "AgroParisTech" affecte la valeur AgroParisTech à la cellule A1.
- La propriété ActiveCell de l'objet Application renvoie un objet Range, qui fait référence à la cellule active de la feuille de calcul. ActiveCell.Font.Bold = True permet d'affecter le style gras à la cellule sélectionnée (Font est une propriété de l'objet Range qui retourne un objet Font contenant les attributs de police (taille, couleur, style, ...) de l'objet Range. Bold est une propriété booléenne de l'objet Font qui correspond au style gras).
- La propriété Selection de l'objet Application renvoie l'objet sélectionné dans la fenêtre active. Le type de l'objet renvoyé dépend de la sélection en cours.

Remarque : La propriété ActiveCell ne permet de faire référence qu'à une seule cellule, alors que la propriété Selection permet de faire référence à une plage de cellules.

V.6. EXERCICES

Ecrire l'instruction qui permet d'affecter la valeur de la cellule B2 de la deuxième feuille de calcul du classeur TEST-MACRO.XLS à la cellule active.

Ecrire une macro qui permet de calculer la surface d'un rectangle (hauteur*largeur) à partir des valeurs contenues dans les cellules A1 et B1 de la deuxième feuille de calcul. Le résultat sera écrit dans la cellule C1.

V.7. LES METHODES DES OBJETS

Les **méthodes** représentent les actions qui peuvent être effectuées par un objet ou que l'on peut appliquer à un objet.

Pour faire référence à une méthode d'un objet donné, il faut utiliser la syntaxe Objet.Méthode.

Exemple :
La méthode Select de l'objet Range permet de sélectionner une cellule ou une plage de cellules.
Range("A1").Select sélectionne la cellule A1.

V.8. EXERCICES

Expliquer ce que fait la macro suivante ?

```
Sub MacroSelection()
   Range("A1:B6").Select
   Selection.Font.Bold = True
   Selection.Value = "Bonjour"
End Sub
```

Compléter cette macro pour que Bonsoir soit écrit dans la cellule B6.

V.9. LES EVENEMENTS

Un **événement** est une action reconnue par un objet. La reconnaissance d'un événement par un objet permet de déclencher l'exécution d'une procédure lorsque cet événement survient. Un clic souris ou la frappe d'une touche au clavier sont des exemples d'événements qui peuvent être interprétés par du code VBA.

Nous verrons plus loin que les contrôles placés dans une boîte de dialogue (boutons de commande, zones de texte, ...) peuvent répondre à des événements. Les classeurs et les feuilles de calcul peuvent également répondre à des événements.

Pour qu'un objet réponde à un événement, il faut écrire du code VBA dans la procédure associée à l'événement considéré.

Ecrire une procédure qui détecte chaque nouvelle plage de cellules ou cellule sélectionnée par l'utilisateur dans la feuille de calcul FEUIL1 du classeur TEST-MACRO.XLS et colorie son fond en bleu.

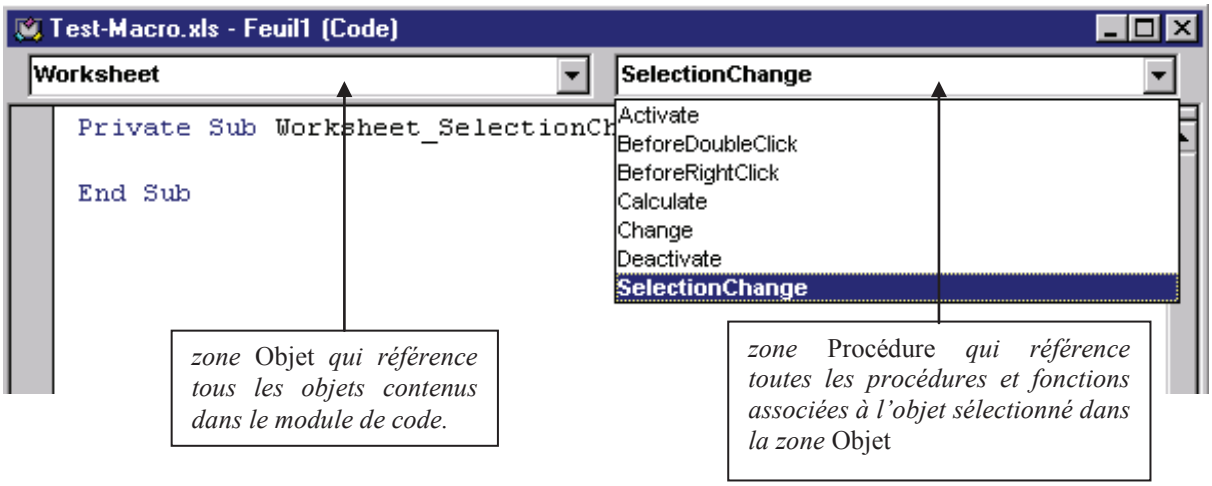
Avant d'écrire cette procédure, il faut se poser plusieurs questions :

- identifier l'objet dont on veut traiter l'événement. Dans notre cas, il s'agit de la feuille de calcul FEUIL1 qui est un objet Worksheet ;
- identifier l'événement à traiter dans la liste des événements associés à l'objet considéré. Dans notre cas, on s'intéresse à chaque nouvelle sélection d'une plage de cellules ou cellule dans la feuille de calcul, ce qui correspond à l'événement SelectionChange ;
- écrire le code VBA approprié dans la procédure associée à l'événement choisi.

1) Ouvrir le classeur TEST-MACRO.XLS, puis l'éditeur de Visual Basic (activer la commande AFFICHAGE BARRE D'OUTILS VISUAL BASIC et cliquer sur l'objet VISUAL BASIC EDITOR).

2) Dans l'explorateur de projets, selectionner la feuille de calcul FEUIL1 et ouvrir son module de code à l'aide du clic droit de la souris et de l'option CODE.

3) Dans la liste de gauche au sommet du module de code, sélectionner l'option WORKSHEET. Dans la liste de droite au sommet du module de code, sélectionner l'événement SELECTIONCHANGE.



On voit alors apparaître dans le module de code l'en-tête de procédure suivant :
Private Sub Worksheet_SelectionChange(ByVal Target As Range)

On veut à présent écrire les instructions VBA permettant de colorier le fond d'une cellule ou d'une plage de cellules en bleue. Pour trouver ces instructions, il existe deux possibilités :

- utiliser l'enregistreur de macro,
- ou explorer les propriétés de l'objet Range pour trouver comment modifier sa couleur de fond.

Le moyen le plus facile pour découvrir les instructions nécessaires pour accomplir une opération donnée consiste à utiliser l'enregistreur de macro.

4) Aller dans EXCEL et **enregistrer une nouvelle macro** MacroCouleurFond qui permet de modifier la couleur de fond d'une cellule.

5) Retourner dans l'éditeur de Visual Basic et étudier le code de la macro MacroCouleurFond.

L'instruction Selection.Interior.ColorIndex = 41 semble intéressante puisqu'elle parle de couleur. Pour obtenir de l'aide sur cette instruction, sélectionner dans le code la propriété ColorIndex et appuyer sur la touche F1. L'aide en ligne de l'éditeur de Visual Basic permet d'obtenir les informations suivantes :

- La propriété Interior d'un objet Range renvoie un objet Interior qui correspond à la couleur de fond d'une plage de cellules.
- La propriété ColorIndex d'un objet Interior permet de déterminer la couleur de fond.

Pour obtenir de l'aide sur un objet, une propriété ou une méthode donnée, sélectionner dans le code cet objet, cette propriété ou cette méthode et appuyer sur la touche F1.

6) Retourner dans le module de code associé à la feuille de calcul FEUIL1 et compléter la procédure Worksheet_SelectionChange comme suit :

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
    Target.Interior.ColorIndex = 41
End Sub
```

La variable Target qui est un objet Range représente la plage de cellules ou cellule sélectionnée par l'utilisateur dans la feuille de calcul FEUIL1.

7) Enregistrer la procédure et retourner dans EXCEL pour la tester (sélectionner une plage de cellules ou cellule et vérifier que sa couleur de fond devient bleue).

V.10. EXERCICE

Ecrire une procédure qui affiche Bonjour ! dans la cellule active avec la police suivante : taille 12, style italique, couleur rouge, à chaque fois que l'utilisateur active la feuille de calcul FEUIL2.

Avant d'écrire cette procédure, il faut se poser plusieurs questions :

- identifier l'objet dont on veut traiter l'événement ;
- identifier l'événement à traiter dans la liste des événements associés à l'objet considéré ;
- écrire le code VBA approprié dans la procédure associée à l'événement choisi.

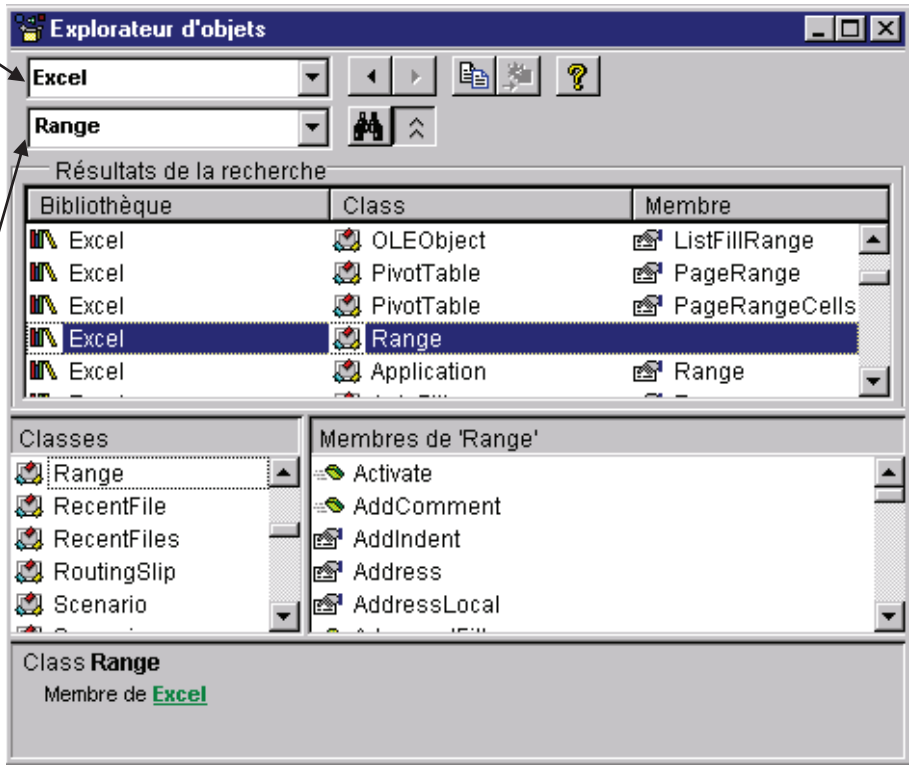
V.11. L'EXPLORATEUR D'OBJETS

L'explorateur d'objets recense l'ensemble des objets disponible dans VBA, leurs propriétés, leurs méthodes et les événements qui leur sont associés.

On appelle **membres** d'un objet ses propriétés, méthodes et événements associés.

Pour ouvrir l'explorateur d'objets, aller dans l'éditeur de Visual Basic et activer la commande AFFICHAGE EXPLORATEUR D'OBJETS.

La zone PROJET/BIBLIOTHEQUE permet de sélectionner le projet ou la bibliothèque d'objets de son choix. Vous pouvez ne sélectionner que la bibliothèque EXCEL.



La zone RECHERCHER TEXTE permet de rechercher un élément (objet ou membre) dans l'explorateur d'objets. Taper l'élément recherché et appuyer sur la touche ENTREE. Le résultat de la recherche s'affiche dans la zone RESULTATS DE LA RECHERCHE.

Double cliquer sur l'objet voulu dans la colonne CLASS (le membre dans la colonne MEMBRE) de la zone RESULTATS DE LA RECHERCHE, pour le voir apparaître dans la zone CLASSES (la zone MEMBRES) située en dessous.

La zone CLASSES affiche l'ensemble des objets disponibles dans la bibliothèque sélectionnée.

La zone MEMBRES contient l'ensemble des membres de l'objet sélectionné dans la zone CLASSES. Les membres d'un objet sont par défaut classés par ordre alphabétique. Pour les classer par catégorie (propriétés, méthodes et événements), cliquer sur le bouton droit de la souris dans la zone MEMBRES et choisir l'option MEMBRES DU GROUPE.

Des icônes permettent de distinguer les objets, les propriétés, les méthodes et les événements.

 icône désignant un objet

 icône désignant une propriété

 icône désignant une méthode

 icône désignant un événement

La zone DETAILS située en dessous de la zone CLASSES affiche un bref descriptif de l'élément sélectionné dans la zone CLASSES ou dans la zone MEMBRES.

Pour obtenir de l'aide sur un objet ou un membre, sélectionner cet objet dans la zone CLASSES ou ce membre dans la zone MEMBRES et cliquer sur la touche **F1**.

V.12. EXERCICE

Rechercher la fonction qui permet de tester si une cellule est vide (rechercher « empty » dans la bibliothèque VBA).

Ecrire une procédure qui colorie une cellule en rouge si elle est vide et en bleu sinon à chaque fois que l'utilisateur double clic sur une cellule du classeur. Vous pourrez utiliser la structure de bloc If (cf. page 43)

```
If condition Then
        Instructions_si_vrai
[Else
        Instructions_si_faux]
End If
```

```
If condition_1 Then
        Instructions_1
ElseIf condition_2 Then
        Instructions_2
ElseIf condition_3 Then
        Instructions_3
        ...
Else
        Instructions_n
End If
```

V.13. COMMENT OBTENIR DE L'AIDE : QUELQUES ASTUCES

Utiliser l'**enregistreur de macro** pour découvrir les instructions VBA nécessaires pour accomplir une opération donnée.

Utiliser l'**explorateur d'objets** pour découvrir les objets disponibles dans VBA et les propriétés, méthodes et événements qui leur sont associés.

Utiliser la **touche F1** pour obtenir de l'aide sur un objet, une propriété, une méthode ou un événement, dans un module de code ou dans l'explorateur d'objets.

VI. LES FONCTIONS DEFINIES PAR L'UTILISATEUR

VBA offre la possibilité de créer ses propres fonctions, qui peuvent être utilisées dans EXCEL comme n'importe quelle fonction intégrée.

Une **fonction** est une suite d'instructions qui retourne une valeur. Elle commence par le mot clé Function suivi du nom de la fonction et d'une liste d'arguments entre parenthèses, qui peut être vide. Elle se termine par le mot clé End Function. Une fonction a la syntaxe suivante :

```
Function NomFonction([argument_1,..., argument_n])
   Instructions
   ...
   NomFonction = Expression          'valeur de retour
   ...
End Function
```

Une fonction indique la valeur à retourner en initialisant son nom avec la valeur de retour.

Remarque : Les crochets **[ ]** signifient que les arguments sont facultatifs.

Attention : Dans VBA, les arguments d'une fonction sont séparés par des virgules, alors que dans EXCEL ils sont séparés par des points-virgules.

VI.1. CREER UNE FONCTION PERSONNALISEE

Ecrire une fonction qui calcule la surface d'un cercle à partir de son rayon.

1) Ouvrir le classeur TEST-MACRO.XLS, puis l'éditeur de Visual Basic.

2) Insérer un nouveau module de code MODULE2 et écrire la fonction suivante :

```
Function SurfaceCercle(Rayon As Double) As Double
  SurfaceCercle = WorksheetFunction.Pi() * Rayon * Rayon
End Function
```

SurfaceCercle est une fonction à un argument : le rayon du cercle Rayon, qui est une variable de type réel. Elle renvoie la surface du cercle de rayon Rayon, qui est de type réel. Pi est une fonction intégrée d'EXCEL.

<u>Attention</u> : Dans VBA les réels sont écrits avec des points, alors qu'ils sont écrits avec des virgules dans EXCEL.

3) Retourner dans EXCEL et saisir les données suivantes dans la feuille de calcul FEUIL1 :

	A	B
5	**Rayon**	**Surface**
6	3	
7	4,5	
8	6	

4) Dans la cellule B6, saisir la formule =SurfaceCercle(A6).

	A	B
5	**Rayon**	**Surface**
6	3	28,26
7	4,5	63,585
8	6	113,04

Lorsqu'on écrit =SurfaceCercle(A6) dans la cellule B6 la valeur de la cellule A6 est transmise à la fonction SurfaceCercle par le biais de l'argument Rayon. La fonction SurfaceCercle initialise alors son propre nom avec le résultat calculé (WorksheetFunction.Pi() * Rayon * Rayon). Le résultat est ensuite retourné à la cellule B6.

5) Recopier la formule =SurfaceCercle(A6) dans les cellules B7 et B8.

La fonction SurfaceCercle peut être utilisée comme n'importe quelle fonction intégrée d'EXCEL (commande INSERTION FONCTION... puis CATEGORIE DE FONCTIONS : PERSONNALISEES). Par contre, elle n'apparaît pas dans la liste des macros d'EXCEL (commande OUTILS MACRO MACROS).

VI.2. EXERCICE

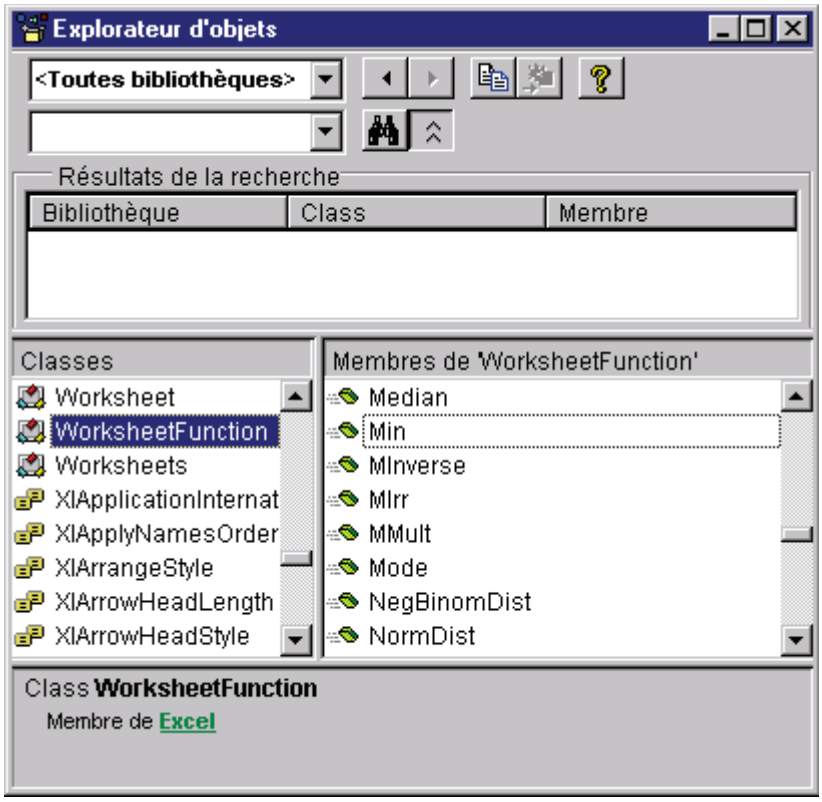
Ecrire une fonction qui permet de calculer la surface d'un rectangle (hauteur*largeur) et la tester dans EXCEL.

VI.3. UTILISER LES FONCTIONS INTEGREES D'EXCEL

Pour utiliser une fonction intégrée d'EXCEL dans VBA, il faut préciser à VBA où peut être trouvée cette fonction. Pour cela, il faut faire précéder le nom de la fonction de l'objet WorkSheetFunction.

Attention : Toutes les fonctions intégrées d'EXCEL ne sont pas disponibles dans VBA !

Pour connaître les fonctions intégrées d'EXCEL disponibles dans VBA, activer la commande AFFICHAGE EXPLORATEUR D'OBJETS dans l'éditeur de Visual Basic et rechercher les membres de l'objet WorksheetFunction.



Attention : Il est fortement conseillé d'utiliser une fonction intégrée d'EXCEL pour faire un calcul donné, si cette dernière existe, plutôt que de définir sa propre fonction, qui serait moins efficace en temps de calcul.

VI.4. EXERCICE

Ecrire une fonction permettant de calculer l'écart entre la valeur maximale et la valeur minimale d'une plage de cellules passée en argument à l'aide des fonctions intégrées d'EXCEL Max et Min et la tester dans EXCEL.

VII.LE LANGAGE VBA

Nous présentons dans ce chapitre les éléments du langage VBA qui sont communs à toutes les applications MICROSOFT OFFICE et plus largement aux langages de programmation, même si la syntaxe et les mots clés varient d'un langage à l'autre.

VII.1. LES VARIABLES ET LES CONSTANTES

La déclaration des variables

Une variable permet de stocker une valeur pouvant être modifiée au cours de l'exécution d'un programme.

Le mot clé Dim permet de **déclarer** explicitement une variable :
Dim NomVariable
où NomVariable est une suite de caractères formés avec des lettres, des chiffres et le caractère souligné _. Le premier caractère est obligatoirement une lettre. Les minuscules sont distinguées des majuscules.

Le langage VBA permet de ne pas déclarer explicitement les variables. Un des intérêts de la déclaration explicite des variables est d'éviter les erreurs de frappes malheureusement très fréquentes en programmation.

Exemple :
```
Sub MacroTestVariable()
   valeur = Range("A1").value
   If valleur >= 18 And valleur <= 20 Then Range("B1").value = "OK"
End Sub
```
Quelque soit la valeur de la cellule A1, cette macro n'écrira pas "OK" dans la cellule B1. Pourtant VBA ne détecte aucune erreur dans le code.

Il est donc préférable de forcer la déclaration explicite des variables en VBA. Pour ce faire, il suffit de placer l'instruction Option Explicit en haut des modules de code avant toutes procédures et toutes fonctions.

```
Option Explicit

Sub MacroTestVariable()
   Dim valeur
   valeur = Range("A1").value
   If valleur >= 18 And valleur <= 20 Then Range("B1").value = "OK"
End Sub
```

VBA détecte une erreur à la compilation : la variable **valleur** n'est pas définie.

Pour ajouter automatiquement l'instruction Option Explicit à tous les nouveaux modules de code créés, activer la commande OUTILS OPTIONS..., cliquer sur l'onglet EDITEUR et cocher la case DECLARATION DES VARIABLES OBLIGATOIRE.

Le type des variables

La plupart des langages de programmation imposent de déterminer le type de données qui peut être stockée dans une variable lors de sa déclaration. En VBA, ce n'est pas obligatoire.

Par défaut, une variable non typée est du type Variant, qui permet de stocker n'importe quel type de données.

Dans un souci d'efficacité du code, il est préférable de typer ses variables. Une variable de type Variant prend, en effet, plus de mémoire que n'importe quel autre type et utilise plus de ressource système pour son traitement (identification du type effectif de la variable et éventuellement conversion).

La déclaration du type d'une variable se fait comme suit :
Dim NomVariable As Type où Type détermine le type de la valeur qui peut être stockée dans la variable.
Les différents types de données disponibles en VBA sont les suivants (cf. aide en ligne de VBA, commande ? SOMMAIRE ET INDEX, onglet SOMMAIRE, option REFERENCE DU LANGAGE VISUAL BASIC) :

- Byte, Integer et Long pour les entiers ;
- Single, Double et Currency pour les réels ;
- Boolean pour les booléens (True ou False) ;
- String pour les chaînes de caractères ;
- Date pour les dates ;
- Object pour faire référence à un objet ;
- Variant pour n'importe quel type.

Exemple :

```
Dim Nom As String
Nom = "Jean " & "Dupond"          'l'opérateur & permet de concaténer des chaînes de caractères

Dim Age As Byte
Age = 23

Dim MaPlage As Object             'variable de type Object
Set MaPlage = Range("A1:B5")      'l'instruction Set attribue une référence d'objet à une variable

Dim MaFeuille As Worksheet        'variable de type Worksheet
Set MaFeuille = ThisWorkbook.Sheets(1)
```

Remarque : On peut typer les arguments d'une procédure ou d'une fonction et la valeur de retour d'une fonction.

Exemple :

```
Function SurfaceCercle (Rayon As Double) As Double
   SurfaceCercle = WorksheetFunction.Pi() * Rayon * Rayon
End Function
```

Remarque sur l'initialisation des variables : Selon leur type, les variables ne sont pas initialisées avec la même valeur :

- les variables de type Byte, Integer ou Long sont initialisées à 0 ;
- les variables de type Single, Double ou Currency sont initialisées à 0 ;
- les variables de type Boolean sont initialisées à False ;
- les variables de type String sont initialisées à "" (la chaîne vide) ;
- les variables de type Date sont initialisées à 00:00:00 ;
- les variables de type Variant sont initialisées à "" (la chaîne vide) (il en est de même pour les variables déclarées implicitement).

Exercice

Soit la plage de cellule A1:B5 contenant dans la colonne A les notes d'un étudiant dans cinq modules différents et dans la colonne B le coefficient de chaque module. Ecrivez une procédure qui permet de calculer la moyenne de l'étudiant et affiche le résultat dans la cellule C6.

La portée des variables

La portée d'une variable définit quelles procédures ou fonctions peuvent utiliser cette variable.

Les variables déclarées à l'intérieur d'une procédure ou d'une fonction ne sont accessibles qu'à l'intérieur de cette procédure ou de cette fonction.
Exemple :

```
Sub InitialiseAge()
   Dim Age As Integer
   Age = 25
End Sub
```

La variable Age n'est accessible que dans la procédure InitialiseAge.

Pour qu'une variable soit accessible à l'ensemble des procédures et des fonctions d'un module, elle doit être déclarée **au début du module** (première instruction après l'instruction Option Explicit) à l'extérieur de toute procédure et de toute fonction.

Pour qu'une variable soit accessible à l'ensemble des procédures et des fonctions d'un projet, elle doit être déclarée au début d'un module à l'extérieur de toute procédure et de toute fonction, à l'aide du mot clé Public.
Exemple :

```
Public Age As Integer
Sub ModifieAge()
   Age = 27
End Sub
```

Exercice

Ecrire les procédures InitialiseAge(), ModifieAge() et AfficheAge() qui affiche l'âge dans la cellule A1. Puis les tester sous Excel à l'aide d'une nouvelle procédure LancerAge() qui permet de les exécuter les unes à la suite des autres (cf. VII.4 page 41 sur les appels de procédures et fonctions) :

```
Sub LancerAge()
   Call InitialiseAge
   Call ModifieAge
   Call AfficheAge
End Sub
```

La déclaration des constantes

Une constante permet d'attribuer un nom à une valeur fixe. La déclaration d'une constante se fait à l'aide du mot clé Const comme suit :

```
Const NomConstante [As Type] = valeur
```

Une constante a la même portée qu'une variable.

Exemple :

```
Public Const Pi As Single = 3.14159265358979
Function SurfaceCercle (Rayon As Double) As Double
    SurfaceCercle = Pi * Rayon * Rayon
End Function
```

L'instruction With...End With

L'instruction With...End With est utile lorsque les références à un objet sont répétées plusieurs fois dans une petite section de code.

Exemple :

```
Sub ModifiePolice(MaPlage As Range)
   MaPlage.Select
   Selection.Font.Size = 12
   Selection.Font.ColorIndex = 3
   Selection.Font.Italic = True
End Sub
```

La macro ModifiePolice peut être réécrite plus simplement comme suit :

```
Sub ModifiePolice(MaPlage As Range)
   MaPlage.Select
   With Selection.Font
      .Size = 12
      .ColorIndex = 3
      .Italic = True
   End With
End Sub
```

Tout ce qui commence par un point dans un bloc d'instructions With...End With est une propriété ou une méthode de l'objet qui suit l'instruction With.

Exercice

En s'aidant éventuellement de l'aide en ligne de l'éditeur de Visual Basic, expliquer ce que fait cette procédure.

```
Private Sub Workbook_SheetActivate(ByVal Sh As Object)
   Dim NbFeuilles As Integer
   Dim n As Integer
   Dim nom As String
   Dim chaine As String
   NbFeuilles = ThisWorkbook.Sheets.Count
   n = Sh.Index
   nom = Sh.Name
   chaine = "Feuille " & n & "/" & NbFeuilles & " : " & nom
   Range("A1").Value = chaine
End Sub
```

VII.2. EXECUTER UNE PROCEDURE SANS ARGUMENT

<u>Attention</u> : Seules les procédures sans argument peuvent être exécutées directement !

Les procédures avec arguments et les fonctions ne peuvent être exécutées que par d'autres procédures ou fonctions et n'apparaissent pas dans la liste des macros de la boîte de dialogue MACRO (commande OUTILS MACRO MACROS de EXCEL).

Pour exécuter une procédure sans argument, positionner le curseur dessus et activer la commande EXECUTION EXECUTER SUB/USERFORM ou cliquer sur l'icône 

Pour arrêter l'exécution d'une procédure sans argument si une erreur survient, utiliser la commande EXECUTION REINITIALISER ou cliquer sur l'icône 

VII.3. LES ENTREES ET SORTIES STANDARDS

Nous présentons deux fonctions d'entrées/sorties standards de VBA : la fonction MsgBox et la fonction InputBox.

La fonction MsgBox

La fonction MsgBox affiche un message dans une boîte de dialogue, attend que l'utilisateur clique sur un bouton, puis renvoie un entier indiquant le bouton choisi par l'utilisateur. Elle a la syntaxe suivante (les arguments entre crochets sont facultatifs) :

```
MsgBox(prompt[,buttons][,title][,helpfile, context])
```

où prompt est le message affiché dans la boîte de dialogue, buttons correspond au type de boutons de la boîte de dialogue et title est le titre de la boîte de dialogue (cf. aide en ligne de l'éditeur de Visual Basic à l'aide de la touche F1.).

1) Ouvrir le classeur TEST-MACRO.XLS, puis l'éditeur de Visual Basic.

2) Insérer un nouveau module de code à l'aide de la commande INSERTION MODULE et saisir la macro suivante :

```
Sub MacroTestMessage()
   Call MsgBox("Bonjour ! ", , "Accueil")
End Sub
```

3) Exécuter la macro à l'aide de la commande EXECUTION EXECUTER SUB/USERFORM ou cliquer sur l'icône 

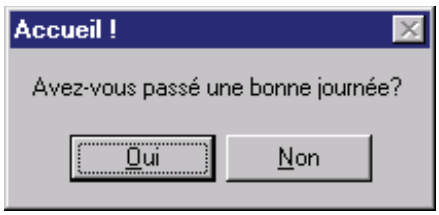
Les valeurs des arguments de la fonction MsgBox peuvent être spécifiées de deux manières : par **position** ou par **nom**.

Exemple :
Dans l'instruction Call MsgBox("Bonjour ! ", , "Accueil"), les valeurs des arguments sont spécifiées par position : les arguments doivent alors être saisis dans le bon ordre et des virgules supplémentaires doivent être inclues pour les arguments manquants.
Cette instruction peut être réécrite en spécifiant les valeurs des arguments par nom comme suit : Call MsgBox(prompt:="Bonjour ! ", Title:="Accueil")
L'ordre entre les arguments n'a alors plus d'importance et il n'y a pas besoin de virgules supplémentaires pour les arguments manquants.

Il existe deux manières de spécifier les valeurs des arguments d'une fonction : par position ou par nom. Lorsque les valeurs des arguments sont spécifiées par nom, il faut utiliser le signe := entre le nom de l'argument et sa valeur.

4) Saisir la macro suivante et l'exécuter.

```
Sub MacroTestMessage2()
   Call MsgBox(prompt:="Avez-vous passé une bonne journée? ", Buttons:=4, Title:="Accueil !")
End Sub
```



L'argument Buttons:=4 permet d'afficher les boutons Oui et Non. Au lieu d'utiliser la valeur numérique 4 pour l'argument Buttons, on aurait pu utiliser la constante associée vbYesNo (cf. aide en ligne de l'éditeur de Visual Basic), qui rend le code plus lisible.

```
Call MsgBox(prompt:="Avez-vous passé une bonne journée? ", Buttons:=vbYesNo, Title:="Accueil !")
```

L'utilisation de constantes VBA intégrées rend le code plus lisible.

Dans l'exemple précédent, une question est posée à l'utilisateur, mais sa réponse n'est pas récupérée : on ne sait pas s'il a passé une bonne journée. Il se pose alors le problème de récupérer la valeur de retour de la fonction MsgBox. Pour ce faire, il ne faut pas utiliser l'instruction Call pour appeler MsgBox.

```
reponse = MsgBox(prompt:="Avez-vous passé une bonne journée?")
```

On peut tester la valeur de retour de la fonction MsgBox (cf. aide en ligne de l'éditeur de Visual Basic), comme suit :

```
Sub MacroTestMessage3()
   Dim reponse As String
   reponse = MsgBox(prompt:="Avez-vous passé une bonne journée? ", Buttons:=vbYesNo, Title:="Accueil !")
   If reponse = vbYes Then
      Call MsgBox("Bonjour ! ", , "Accueil")
   Else
      Call MsgBox("Bonsoir ! ", , "Accueil")
   End If
End Sub
```

La fonction InputBox

La fonction InputBox affiche une invite dans une boîte de dialogue, attend que l'utilisateur tape du texte ou clique sur un bouton, puis renvoie le contenu de la zone de texte sous la forme d'une chaîne de caractère. Elle a la syntaxe suivante (les arguments entre crochets sont facultatifs) :

```
InputBox(prompt[,title][,default][,xpos][,ypos] [,helpfile,context])
```

où prompt est le message affiché dans la boîte de dialogue et title est le titre de la boîte de dialogue (cf. aide en ligne de l'éditeur de Visual Basic).

Si l'utilisateur ne saisit rien dans la zone de texte, la fonction InputBox renvoie la chaîne de caractères vide "".

5) Saisir la macro suivante et l'exécuter.

```
Sub MacroTestDialogue()
    Dim reponse As String
    reponse = InputBox("Quel age avez-vous ?", "Saisie age")
    If reponse = "" Then
        Call MsgBox("Vous n'avez pas répondu à la question !", , "Erreur")
    Else
        Call MsgBox("Vous avez " & reponse & " ans", , "Age")
    End If
End Sub
```

L'opérateur & permet de concaténer des chaînes de caractères. Par exemple, si la réponse donnée par l'utilisateur est 22, alors on a :

```
"Vous avez " & reponse & " ans" = "Vous avez 22 ans"
```

VII.4. RECAPITULATIF SUR LES PARENTHESES ET LES LISTES D'ARGUMENTS

Les procédures et les fonctions peuvent avoir des arguments. Voici un petit récapitulatif sur leur appel avec arguments.

- Pour récupérer la valeur de retour d'une fonction, il ne faut pas utiliser l'instruction Call pour l'appeler.

Exemple : reponse = InputBox("Quel age avez-vous ?", "Saisie age")

- Pour appeler une procédure ou ne pas récupérer la valeur de retour d'une fonction, il faut utiliser l'instruction Call.

Exemple : Call MsgBox("Bonjour ! ", , "Accueil")

Attention : Il ne faut pas mettre d'espace entre le nom d'une fonction ou d'une procédure et la liste de ses arguments placés entre parenthèses.

Il existe un autre moyen pour appeler une procédure ou une fonction dont on ne veut pas récupérer la valeur de retour : ne pas utiliser l'instruction Call et ne pas placer ses arguments entre parenthèses.
Exemple :

```
MsgBox "Bonjour ! ", , "Accueil"
```

Lors de l'appel d'une procédure ou d'une fonction, les valeurs de ses arguments peuvent être spécifiées par **position** ou par **nom**.
Exemple :

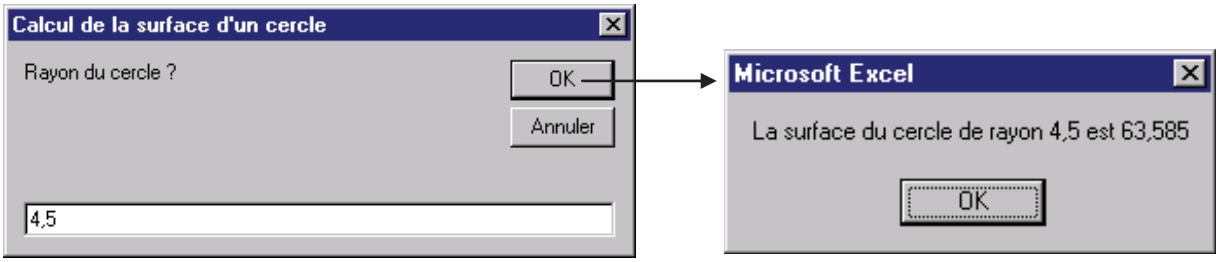
```
'arguments spécifiés par position
Call MsgBox("Bonjour ! ", , "Accueil")

'arguments spécifiés par nom
Call MsgBox(prompt :="Bonjour ! ", Title :="Accueil")
```

VII.5. EXERCICE

Exercice

Ecrire une procédure qui calcule la surface d'un cercle à partir du rayon donné par l'utilisateur. Vous vérifierez que l'utilisateur saisie bien une valeur pour le rayon (réponse différente de la chaine de caractères "") et que la valeur saisie est bien un nombre.



Exercice

En utilisant les procédures InitialiseAge(), ModifieAge(), AfficheAge() et LancerAge() (cf. exercice page 34), écrire une procédure permettant de faire les instructions suivantes :

- Si aucun âge n'a été défini (cellule A1 vide), demander son âge à l'utilisateur et le copier dans la cellule A1 ;
- Sinon, donner l'âge de la cellule A1 à l'utilisateur et lui demander s'il souhaite le modifier
  - Si oui, demander la nouvelle valeur et modifier la cellule A1
  - Sinon sortir.

VII.6. LES ENONCES CONDITIONNELS

VBA permet de traiter les énoncés conditionnels à l'aide de l'instruction If. On distingue l'instruction If écrite sur une seule ligne et le bloc If.

L'instruction If écrite sur une seule ligne

L'instruction If écrite sur une seule ligne a la syntaxe suivante :
If condition **Then** Inst_si_vrai [**Else** Inst_si_faux]

Les sections Then et Else d'une instruction If ne peuvent contenir qu'une seule instruction et doivent être écrites sur une seule ligne.

Exemple :

```
If test = True Then Call MsgBox("Vrai") Else Call MsgBox("Faux")
```

Le bloc If

La structure de bloc If permet aux sections Then et Else de contenir plusieurs instructions. Elle peut avoir l'une des deux syntaxes suivantes :

```
If condition Then
        Instructions_si_vrai
[Else
        Instructions_si_faux]
End If
```

```
If condition_1 Then
        Instructions_1
ElseIf condition_2 Then
        Instructions_2
ElseIf condition_3 Then
        Instructions_3
        ...
Else
        Instructions_n
End If
```

Dans une structure de bloc If, il faut impérativement aller à la ligne après le Then.

Remarque : Les crochets [ ] signifient que le bloc Else est facultatif.

Lorsque le bloc If contient une ou plusieurs sections ElseIf, les différentes conditions sont testées jusqu'à ce qu'une soit évaluée à vraie. Si aucune condition n'est vérifiée, les instructions de la section Else, si elle existe, sont exécutées. Si aucune condition n'est vérifiée et si la section Else n'existe pas, le bloc If ne fait rien.

VBA offre la possibilité d'utiliser les opérateurs logiques suivants : Or, Xor (ou exclusif), And, Not.

Exemple :

```
Sub BonjourHeureIf()
   Dim heure As Integer
   Dim mes As String
   heure = Left(Time, 2)        'récupération de l'heure, 2 caractères les plus à gauche
                                ‘de la valeur de retour de la fonction Time
   If heure > 7 And heure < 12 Then
      mes = "Bonjour"
   ElseIf heure >= 12 And heure < 18 Then
         mes = "Bon après-midi"
      ElseIf heure >= 18 And heure < 22 Then
            mes = "Bonne soirée"
         Else
            mes = "Bonne nuit"
   End If
   Call MsgBox(mes)
End Sub
```

Exercice

Ecrire une procédure qui teste si la cellule active est vide ou non. Si elle n'est pas vide, la procédure affiche un message avec sa valeur.

VII.7. LES ENONCES ITERATIFS

VBA fournit deux structures pour répéter en boucle les mêmes instructions : l'instruction Do...Loop et l'instruction For...Next.

La boucle Do...Loop

La boucle Do...Loop est utilisée lorsque l'on veut que la répétition des instructions s'arrête quand une condition donnée est vérifiée. On distingue plusieurs syntaxes possibles.

```
Do While condition          'Tant que la condition est vraie, les instructions
    Instructions            'sont exécutées
Loop

Do Until condition          'Jusqu'à ce que la condition devienne vraie, les
    Instructions            'instructions sont exécutées
Loop
```

Avec ces deux boucles, les instructions peuvent ne jamais être exécutées. Pour qu'elles soient exécutées au moins une fois, on peut utiliser l'une des deux syntaxes suivantes :

```
Do                          'les instructions sont exécutées tant que la
    Instructions            'condition reste vraie
Loop While condition

Do                          'les instructions sont exécutées jusqu'à ce que la
    Instructions            'condition devienne vraie
Loop Until condition
```

Exercice

Que fait la boucle suivante (la fonction LCase convertit tous les caractères en minuscules) ? La Réécrire avec l'instruction Do...Loop While.

```
Do
  rep = InputBox("Voulez-vous continuer ? (O/N)")
Loop Until LCase(rep) = "n"
```

La boucle For...Next

La boucle For...Next est utilisée lorsque l'on connait à l'avance le nombre de fois où l'on veut que les instructions soient répétées.

```
For compteur = nbdébut To nbfin [Step nbpas]
        Instructions          'instructions exécutées un nombre déterminé de fois
Next compteur
```

Les instructions de la boucle For...Next sont exécutées un nombre déterminé de fois. Lorsque l'option Step n'est pas renseignée, le compteur est incrémenté de 1 à chaque itération et les instructions sont exécutées nbdébut - nbfin fois.

Exemple :

La procédure suivante calcule la somme des n premiers entiers :

```
Sub SommeEntiers()
   Dim somme As Integer
   Dim compteur As Integer
   Dim n As Integer
   n = InputBox("Donner un entier :", "Saisie entier")
   For compteur = 1 To n
      somme = somme + compteur
   Next compteur
   Call MsgBox("Somme des " & n & " premiers entiers : " & somme)
End Sub
```

Le compteur peut être, à chaque itération, incrémenté de la valeur de nbpas.

Exemple :

```
For compteur = 20 To 0 Step -2
   Call MsgBox(compteur)
Next compteur
```

Exercice

Ecrire une procédure qui permet de calculer la surface d'un rectangle tant que les valeurs de sa hauteur et de sa largeur situé dans les colonnes A et B sont connues (les cellules ne sont pas vides). Vous pourrez utiliser l'instruction Range("A" & i) pour parcourir les i lignes de la colonne A.

La boucle For Each...Next

La boucle For Each...Next permet de parcourir les différents objets d'une collection d'objets EXCEL.

Exemple :

```
Sub ColorieCellule()
   Dim MaPlage As Range
   Dim Cellule As Range
   Dim i As Integer
   Set MaPlage = Range("A1:B5")
   For Each Cellule In MaPlage
      Cellule.Interior.ColorIndex = i
      i = i + 1
   Next Cellule
End Sub
```

Dans la bouche For Each...Next c'est le « type » de l'ensemble d'objets après l'instruction In qui détermine le parcours : on parcourt les cellules d'un ensemble de cellule (objet Range), les feuilles d'un ensemble de feuilles (objet Worksheets), …

L'instruction Exit

L'instruction Exit permet de quitter un bloc Do...Loop, For...Next, Function, ou Sub. Sa syntaxe est la suivante :

```
Exit Do
Exit For
Exit Function
Exit Sub
```

Exemple :

La boucle suivante s'arrête lorsque la réponse rep est n ou N :

```
   Do
      Rep = InputBox("Voulez-vous continuer ? (O/N)")
      If LCase(rep) = "n" Then Exit Do
   Loop
```

Exercice

Ecrire la fonction CompteLignes qui permet de compter le nombre de lignes de la plage de cellules MaPlage passée en argument et la tester dans EXCEL.

```
Function CompteLignes(MaPlage As Range) As Integer
```

Vous pourrez utiliser la boucle For Each...Next pour parcourir les lignes. L'instruction MaPlage.Rows renvoie la plage de cellules MaPlage sous la forme d'un ensemble de lignes.

VII.8. LES TABLEAUX

Un tableau est une variable permettant de stocker plusieurs valeurs.

La déclaration d'un tableau précise le nombre d'éléments du tableau et éventuellement leur type :
Dim NomTableau(dimension) [As Type]

On appelle taille d'un tableau, le nombre d'éléments du tableau.

Les éléments d'un tableau sont indexés de 0 à dimension–1. NomTableau(i) désigne le ième élément du tableau de nom NomTableau, i devant être compris entre 0 et dimension–1.

Exemple :

```
Dim TabInt(10) As Integer       'tableau de 10 entiers
TabInt(0)=12                    'accès au premier élément du tableau
TabInt(9)=36                    'accès au dernier élément du tableau
```

Les tableaux sont très utiles pour traiter des listes d'éléments. La fonction Array renvoie une variable de type Variant contenant un tableau.
Exemple :

```
Dim ListeJours As Variant
ListeJours = Array("Lundi", "Mardi", "Mercredi", "Jeudi", "Vendredi", "Samedi", "Dimanche")
```

La fonction UBound renvoie la valeur maximale des index d'un tableau.
Exemple :
UBound(ListeJours) renvoie 6.

Exercice

Ecrire la fonction VerifierJours(Jour As String) qui permet de vérifier que la chaîne de caractères passée en argument est bien un jour de la semaine et qui renvoie un booléen, et la tester dans EXCEL.

VII.9. LE DEBOGAGE

Le **débogage** consiste à régler les erreurs directement liées au code d'un programme. Trois types d'erreur peuvent affecter un programme écrit en VBA :

- des erreurs de compilation qui surviennent lorsque VBA rencontre une instruction qu'il ne reconnaît pas ;
- des erreurs d'exécution ;
- des erreurs logiques : le programme s'exécute mais le résultat obtenu ne correspond pas à celui attendu.

Les erreurs de compilation et d'exécution

La première étape pour tester un programme consiste à **compiler le programme** à l'aide de la commande DEBOGAGE COMPILER VBAPROJECT. Les erreurs de compilation sont mises en évidence. Recommencer la compilation jusqu'à ce que toutes les erreurs de compilation soient corrigées.

Lorsqu'il n'y a plus d'erreurs de compilation, **exécuter le programme** à l'aide de la commande EXECUTION EXECUTER SUB/USERFORM. 

Si une erreur d'exécution est générée, l'instruction coupable est mise en évidence. Corriger l'erreur et **réinitialiser le programme** à l'aide de la commande EXECUTION REINITIALISER. Recommencer l'exécution jusqu'à ce que toutes les erreurs d'exécution soient corrigées. 

Les erreurs Logiques

Il s'agit maintenant de corriger les erreurs logiques qui sont les plus difficiles à repérer. L'éditeur de Visual Basic propose plusieurs outils de débogage dans le menu DEBOGAGE.

1) Ouvrir le classeur ELEVES.XLS.

	A	B	C	D	E	F	G	H
1	**nom**	**prénom**	**ville**	**note écrit**	**note orale**	**moyenne**	**mention**	**lettre**
2	Detila	Anne	Nice	7	9	8	Passable	FX
3	Gautier	Pascale	Nice	13	11	12	AB	D
4								
5								
6	Leforet	Pierre	Nice	12	11	11,5	Passable	E
7	Michelet	Pierre	Nice	14	16	15	B	C
8	Dupond	Damien	Paris	12	7	9,5	Passable	FX
9	Dupond	Marc	Paris	11	13	12	AB	D
10	Gautier	Marie	Paris	7	13	10	Passable	E
11								
12	Gordet	Fabrice	Paris	11	8	9,5	Passable	FX
13								
14								
15	Hillon	Sophie	Paris	14	12	13	AB	D
16	Roulisa	Paul	Paris	15	13	14	B	C
17	Labreau	Emile	Toulouse	9	12	10,5	Passable	E
18	Martin	Jean	Toulouse	14	11	12,5	AB	D
19	Tulipe	Alice	Toulouse	12	8	10	Passable	E

2) Ouvrir l'éditeur de Visual Basic à partir du classeur ELEVES.XLS.

3) Ouvrir un nouveau module de code à l'aide de la commande INSERTION MODULE. Une fenêtre code MODULE1 s'ouvre.

4) Dans le module de code MODULE1, saisir la procédure suivante qui permet de supprimer les lignes vides de la liste de données :

```
Sub SupprLignesVides()
    Dim MaPlage As Range
    Dim Cellule As Range
    Set MaPlage = Range("A1:A19")
    For Each Cellule In MaPlage
        If IsEmpty(Cellule) Then
            Rows(Cellule.Row).Delete        'supprime toute la ligne
        End If
    Next Cellule
End Sub
```

La fonction IsEmpty permet de tester si une cellule est vide.

5) Exécuter la procédure SupprLignesVides. On constate que s'il existe deux lignes vides consécutives, la deuxième ligne n'est pas supprimée.

	A	B	C	D	E	F	G	H
1	**nom**	**prénom**	**ville**	**note écrit**	**note orale**	**moyenne**	**mention**	**lettre**
2	Detila	Anne	Nice	7	9	8	Passable	FX
3	Gautier	Pascale	Nice	13	11	12	AB	D
4								
5	Leforet	Pierre	Nice	12	11	11,5	Passable	E
6	Michelet	Pierre	Nice	14	16	15	B	C
7	Dupond	Damien	Paris	12	7	9,5	Passable	FX
8	Dupond	Marc	Paris	11	13	12	AB	D
9	Gautier	Marie	Paris	7	13	10	Passable	E
10	Gordet	Fabrice	Paris	11	8	9,5	Passable	FX
11								
12	Hillon	Sophie	Paris	14	12	13	AB	D
13	Roulisa	Paul	Paris	15	13	14	B	C
14	Labreau	Emile	Toulouse	9	12	10,5	Passable	E
15	Martin	Jean	Toulouse	14	11	12,5	AB	D
16	Tulipe	Alice	Toulouse	12	8	10	Passable	E

6) Insérer de nouveau des lignes dans la liste de données pour obtenir la même liste de données que celle du point (1).

7) Dans l'éditeur de Visual Basic, espionner la valeur des variables Cellule.Row et Cellule à l'aide de la commande DEBOGAGE AJOUTER UN ESPION. La fenêtre ESPIONS s'ouvre.

8) Réduire la fenêtre de l'éditeur de Visual Basic de façon à voir le classeur EXCEL au second plan.

9) Exécuter pas à pas la procédure SupprLignesVides à l'aide de la commande DEBOGAGE PAS A PAS DETAILLE (touche F8). Observer attentivement l'effet de chaque instruction de la procédure sur le classeur EXCEL et examiner les valeurs prises par les variables Cellule.Row et Cellule.

Valeur de `Cellule.Row`	Valeur de `Cellule`	Action
1	"nom"	
2	"Detila"	
3	"Gautier"	
4	""	Suppression de la ligne 4
5	"Leforet"	
6	"Michelet"	

Lorsque la procédure atteint la cellule A4, la ligne 4 est supprimée. Cela a pour conséquence de décaler toutes les cellules vers le haut. La cellule A5 passe alors en A4, la cellule A6 en A5... La boucle For Each...Next traite ensuite la cellule suivante soit la cellule A5. Le contenu de la cellule précédemment en A5 ne sera donc pas traité, puisque cette cellule est passée en A4.

10) Arrêter l'exécution de la procédure à l'aide de la commande EXECUTION ARRET.

11) Une solution possible au bogue est de dérouler une boucle commençant par la dernière cellule de la liste de données.

```
Sub SupprLignesVidesCorrigée()
   Dim compteur As Integer
   Dim DerLigne As Integer
   Dim MaPlage As Range
   Set MaPlage = Range("A1:A19")
   DerLigne = MaPlage.Rows.Count
   For compteur = DerLigne To 1 Step -1
      If IsEmpty(Range("A" & compteur)) Then Rows(compteur).Delete
   Next compteur
End Sub
```

Pour information

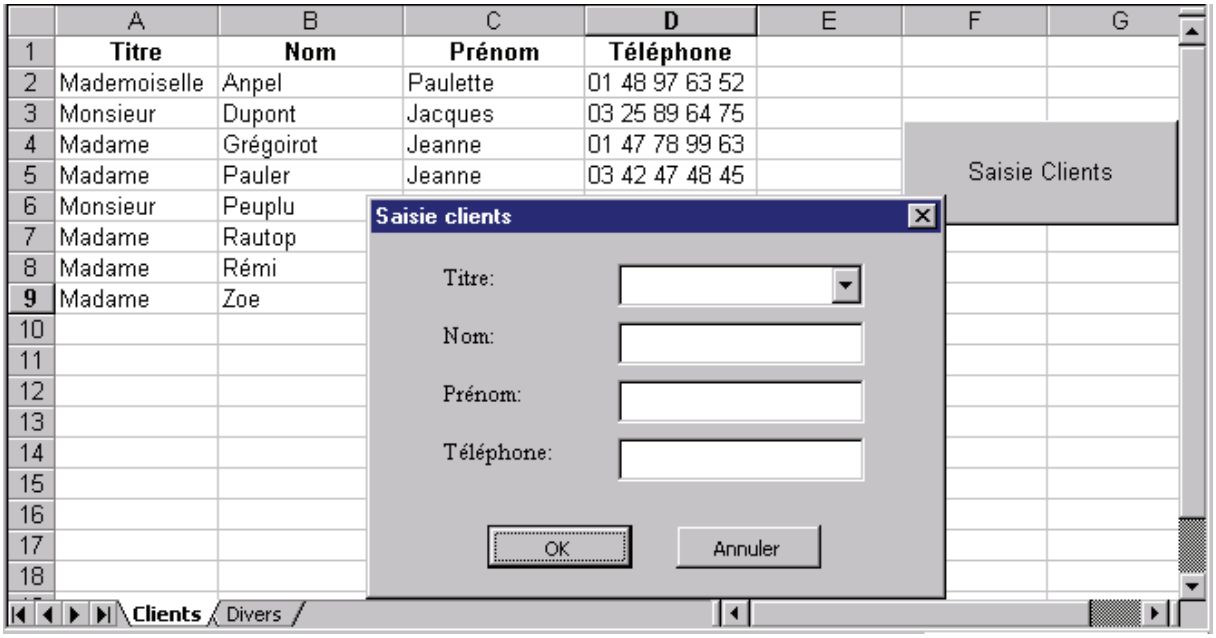
Les principaux outils de débogage proposés par l'éditeur de Visual Basic dans le menu DEBOGAGE sont les suivants.

- **Exécution pas à pas** : pour exécuter un programme pas à pas, placer le curseur dans la procédure à exécuter et activer la commande DEBOGAGE PAS A PAS DETAILLE. L'exécution d'une procédure pas à pas permet d'en examiner le déroulement instruction après instruction. La procédure est dite **en mode arrêt**. On peut à tout moment retourner à un mode d'exécution normal.
- **Les bulles d'aide** : pour visualiser la valeur d'une variable, activer la commande OUTILS OPTIONS, cliquer sur l'onglet EDITEUR et cocher la case INFO-BULLES AUTOMATIQUES. Lorsque le curseur est placé au dessus d'une variable, la valeur de celle-ci apparaît dans une bulle d'aide.
- **La fenêtre Variables locales** : pour visualiser la valeur des variables et constantes aux différents stades de l'exécution pas à pas d'un programme, activer la commande AFFICHAGE FENETRE VARIABLES LOCALES.
- **Les points d'arrêt** : les points d'arrêt permettent d'interrompre l'exécution d'un programme sur une instruction précise. Pour définir un point d'arrêt sur une instruction douteuse, activer la commande DEBOGAGE BASCULER LE POINT D'ARRET.
- **Les espions** : pour espionner la valeur d'une variable, activer la commande DEBOGAGE AJOUTER UN ESPION. Pour afficher la fenêtre ESPIONS, activer la commande AFFICHAGE FENETRE ESPIONS.
- **La pile d'appels** : la pile d'appels recense toutes les procédures ou fonctions en cours d'exécution, selon leur ordre d'appel. Pour afficher la pile des appels, activer la commande AFFICHAGE PILE DES APPELS.

VIII. LES OBJETS USERFORM

Les objets UserForm sont des boîtes de dialogue définies par l'utilisateur.

Ce chapitre explique comment créer une boîte de dialogue permettant de saisir les clients d'une entreprise, comme le montre la capture d'écran suivante :



1) Ouvrir un nouveau classeur et l'enregistrer sous SAISIECLIENTS.XLS.

2) Pour renommer la feuille de calcul FEUIL1, cliquer avec le bouton droit sur l'onglet FEUIL1, activer la commande RENOMMER et taper CLIENTS.

3) Saisir dans la feuille de calcul CLIENTS, les données suivantes :

	A	B	C	D
1	Titre	Nom	Prénom	Téléphone
2				

4) Renommer DIVERS la feuille de calcul FEUIL2 et saisir les données suivantes :

	A
1	Madame
2	Mademoiselle
3	Monsieur

On peut distinguer deux phases dans la création d'un objet UserForm :

- le dessin de l'objet UserForm

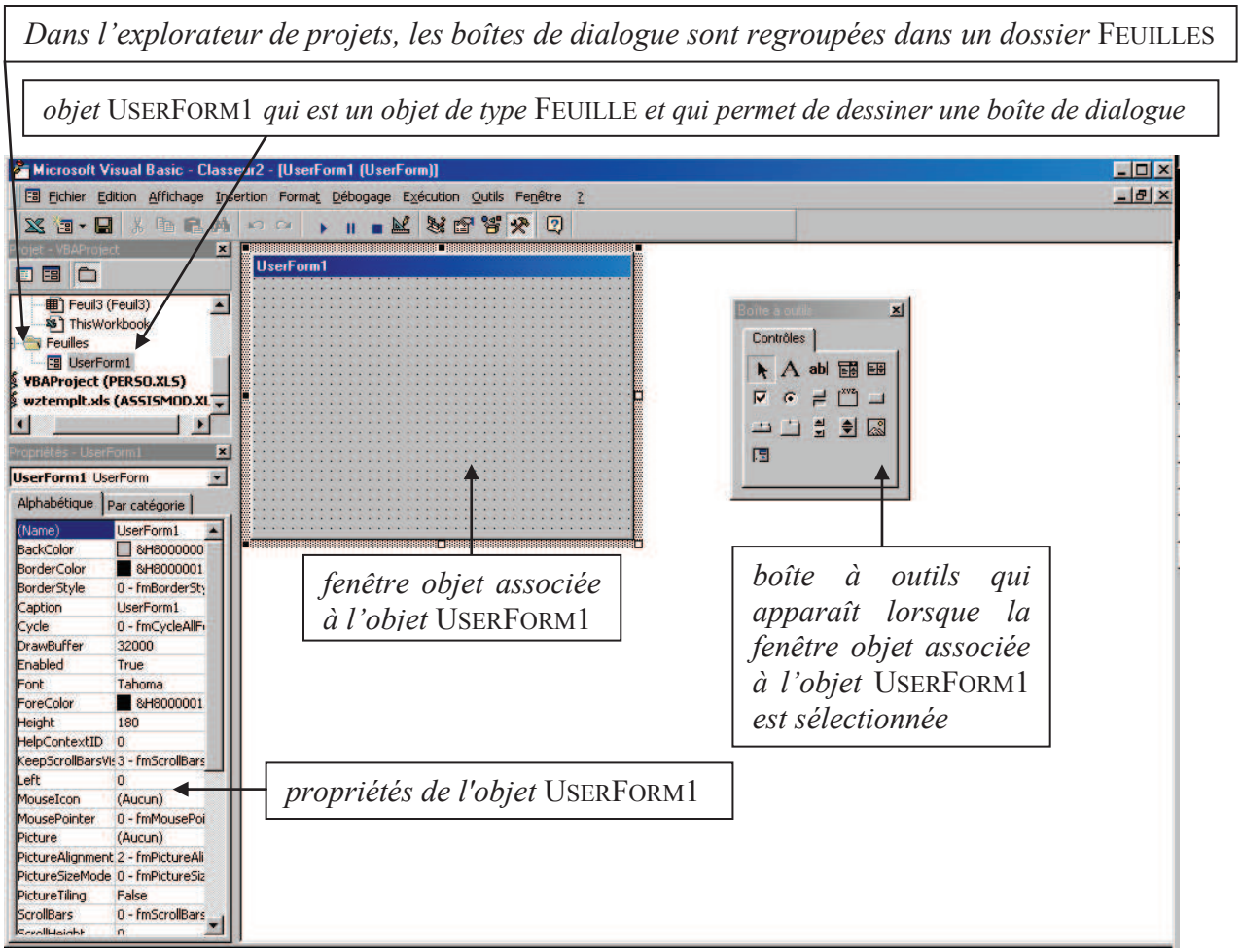
Le dessin d'un objet UserForm consiste à placer des contrôles sur l'objet.
Les **contrôles** sont les éléments constitutifs d'un objet UserForm tels qu'une case à cocher, une zone de texte, une zone de liste modifiable ou un bouton de commande permettant une intervention de l'utilisateur.
Un contrôle est un objet : il possède des propriétés, des méthodes et des événements définis.

- l'association de code à l'objet UserForm et à ses différents contrôles

Il s'agit de déterminer le comportement de l'objet UserForm et de ses contrôles face aux différents événements utilisateur pouvant l'affecter et d'écrire le code permettant d'exploiter les actions effectuées par l'utilisateur.

VIII.1.CREER UN OBJET USERFORM

1) Ouvrir l'éditeur de Visual Basic et activer la commande INSERTION USERFORM. Un objet UserForm1 apparaît, ainsi qu'une boîte à outils permettant d'ajouter des contrôles à l'objet.

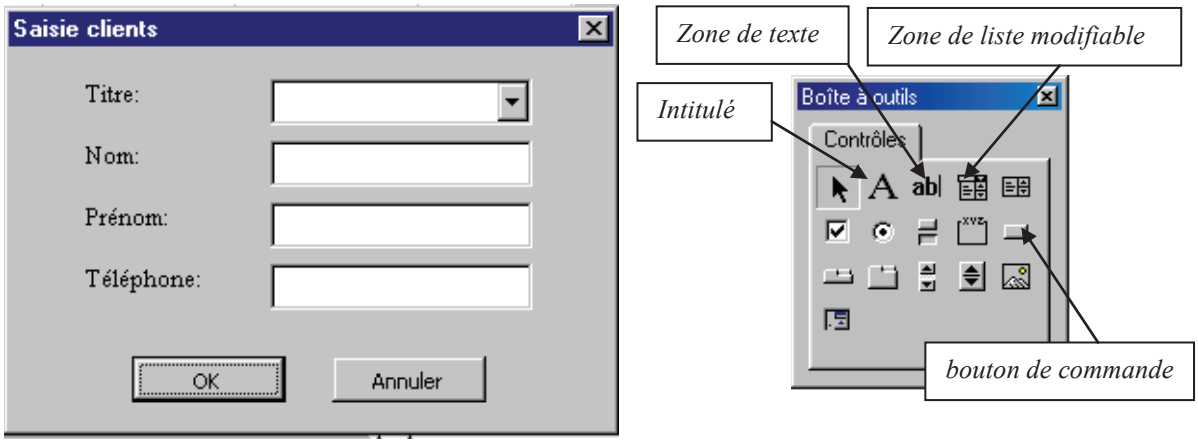


L'explorateur de projets permet d'afficher le code d'un objet USERFORM à l'aide du clic droit de la souris et de l'option CODE ; ou d'afficher l'objet lui-même à l'aide du clic droit de la souris et de l'option AFFICHER L'OBJET.

2) Modifier le nom de l'objet UserForm par défaut, UserForm1, en FenetreSaisieClients dans la propriété NAME de la fenêtre PROPRIETES. Ce nom est le nom qui sera utilisé dans le code pour faire référence à l'objet UserForm.

3) La propriété CAPTION de l'objet UserForm correspond au libellé qui apparaît dans sa barre de titre. Modifier cette propriété en Saisie Clients.

On veut créer l'objet UserForm suivant :



Cet objet est composé des contrôles suivants : un INTITULE et une ZONE DE LISTE MODIFIABLE pou saisir le titre du client (Madame, Mademoiselle ou Monsieur), trois INTITULES et trois ZONES DE TEXTE pour saisir le nom, le prénom et le téléphone du client et deux BOUTONS DE COMMANDE OK et Annuler.

Pour placer un contrôle dans l'objet UserForm, cliquer sur l'objet voulu de la BOITE A OUTILS et le faire glisser vers l'objet UserForm. Dès qu'un contrôle a été placé sur l'objet UserForm, définir son nom grâce à la propriété NAME.

Attention : Il est recommandé d'utiliser des noms évocateurs pour ses contrôles, qui permettent d'identifier le type d'objet qu'ils représentent et leur utilité.

4) Placer les différents contrôles de l'objet UserForm et modifier leurs propriétés comme suit :

Contrôle	propriété NAME	propriété CAPTION	propriété FONT
INTITULE	IntituléTitre	Titre :	police TIMES NEW ROMAN, taille 10
ZONE DE LISTE MODIFIABLE	ComboBoxTitre		"
INTITULE	IntituléNom	Nom :	"
ZONE DE TEXTE	TextBoxNom		"
INTITULE	IntituléPrénom	Prénom :	"
ZONE DE TEXTE	TextBoxPrénom		"
INTITULE	IntituléTél	Téléphone :	"
ZONE DE TEXTE	TextBoxTél		"
BOUTON DE COMMANDE	ButtonOK	OK	
BOUTON DE COMMANDE	ButtonAnnuler	Annuler	

Remarque : Il est possible de copier/coller des contrôles.

VIII.2. AFFICHER ET FERMER UN OBJET USERFORM

Afficher un objet UserForm

L'instruction Load permet de charger un objet UserForm en mémoire sans l'afficher. La méthode Show de l'objet UserForm permet d'afficher un objet UserForm et de le charger en mémoire, si cela n'a pas déjà été fait.

Ecrire la procédure qui permet d'afficher la boîte de dialogue de saisie d'un client.

1) Dans l'éditeur de Visual Basic, activer la commande INSERTION MODULE. Un module de code MODULE1 s'ouvre. Renommer le ModuleSaisieClients à l'aide de la propriété NAME.

2) Saisir la procédure suivante dans le module de code :

```
Sub ProgPrincSaisieClients()
   Sheets("Clients").Activate          'activation de la feuille Clients
   FenetreSaisieClients.Show
End Sub
```

3) Exécuter cette procédure (pour fermer la boîte de dialogue FenetreSaisieClients, cliquer sur son bouton de fermeture 5).

Fermer ou masquer un objet `UserForm`

L'instruction Unload permet de fermer un objet UserForm et de l'effacer de la mémoire, les valeurs de ses contrôles sont alors perdues. La méthode Hide de l'objet UserForm permet de faire disparaître un objet UserForm de l'écran sans le supprimer de la mémoire.

L'instruction Unload ou la méthode Hide sont généralement placées dans les procédures événementielles attachées aux boutons de validation de l'objet UserForm, comme par exemple les boutons de commande OK et Annuler.

VIII.3. ASSOCIER DU CODE A UN OBJET USERFORM

Les contrôles placés sur un objet UserForm et l'objet UserForm lui même sont réceptifs aux événements utilisateurs qui les affectent (clic souris sur un bouton de commande, saisie d'une valeur dans une zone de texte…). On peut ainsi créer des procédures dites **procédures événementielles**, qui se déclencheront lorsque l'événement correspondant sera repéré.

La syntaxe d'une procédure événementielle attachée à un contrôle de nom NomContrôle (propriété NAME) et déclenchée par un événement NomEvénement est la suivante :

```
Private Sub NomContrôle_NomEvénement()
    ...
End Sub
```

Dans le cas d'une procédure événementielle attachée à un objet UserForm, le nom de l'objet UserForm (propriété NAME) n'apparaît pas dans les instructions de déclaration de la procédure. Il est remplacé par le mot clé UserForm comme suit :

```
Private Sub UserForm_NomÉvénement ()
    ...
End Sub
```

Les **événements** sont nombreux et varient d'un contrôle à l'autre. En voici, quelques uns :

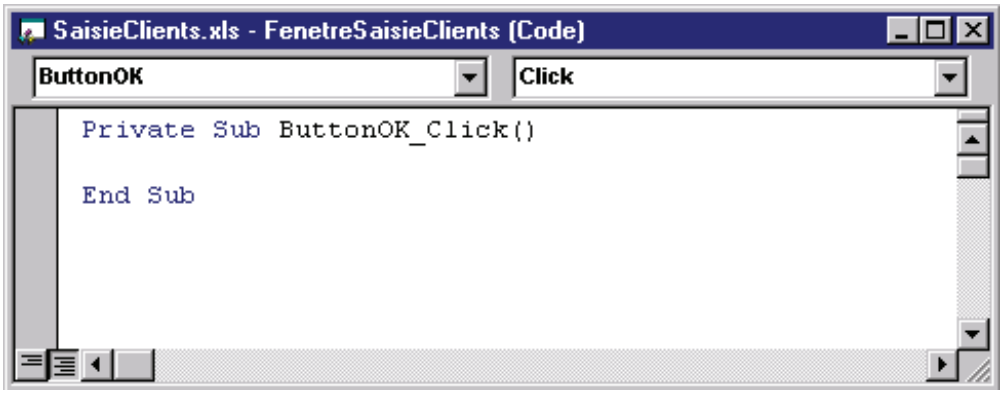
- Événement Change : détecté lors de la modification de la valeur (propriété VALUE) d'un contrôle (par exemple une zone de texte).
- Événement Click : détecté lorsque l'utilisateur clique sur un contrôle (par exemple un bouton de commande).
- Événement dblClick : détecté lorsque l'utilisateur double-clique sur un contrôle.

Associer du code à un bouton de commande

Associer le code nécessaire au bouton de commande OK pour que la boîte de dialogue soit fermée lorsque l'on clique dessus.

1) Aller dans l'éditeur de Visual Basic. Dans l'explorateur de projets, sélectionner l'objet UserForm de nom FenetreSaisieClients et ouvrir son module de code à l'aide du clic droit de la souris et de l'option CODE (ou en double-cliquant dessus).

2) Dans la liste de gauche au sommet du module de code, sélectionner l'option ButtonOK. Dans la liste de droite au sommet du module de code, sélectionner l'événement Click.



3) Compléter la procédure événementielle ButtonOK_Click qui apparaît comme suit :

```
Private Sub ButtonOK_Click()
   Call Unload(Me)
End Sub
```

La propriété Me de l'objet UserForm renvoie l'objet UserForm actif.

4) Pour tester cette procédure, exécuter la procédure ProgPrincSaisieClients du module de code ModuleSaisieClients.

Exercice

Associer le code nécessaire au bouton Annuler pour que la boîte de dialogue soit fermée lorsque l'on clique dessus.

Initialiser un objet UserForm

L'événement Initialize d'un objet UserForm est détecté lorsque l'objet UserForm est chargé, à l'aide de la méthode Show ou de l'instruction Load.

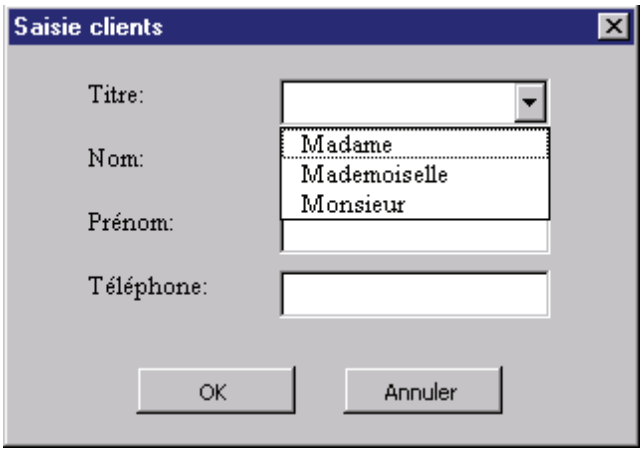
Attention : Lorsque la méthode Show est appliquée à un objet UserForm masqué par la méthode Hide, l'objet UserForm n'est pas rechargé mais uniquement affiché. L'événement Initialize n'est alors pas reconnu.

La procédure événementielle associée à l'événement Initialize d'un objet UserForm s'exécute avant l'affichage de l'objet et a la syntaxe suivante :

```
Private Sub UserForm_Initialize()
    ...
End Sub
```

Cette procédure permet d'effectuer des réglages dans l'objet UserForm avant son chargement.

Ecrire le code nécessaire pour affecter les valeurs "Madame", "Mademoiselle" et "Monsieur" à la zone de liste modifiable ComboBoxTitre, en utilisant les valeurs qui se trouvent dans la plage de cellule A1:A3 de la feuillle de calcul DIVERS.



1) Aller dans l'éditeur de Visual Basic. Dans l'explorateur de projets, selectionner l'objet UserForm de nom FenetreSaisieClients et ouvrir son module de code à l'aide du clic droit de la souris et de l'option CODE.

2) Dans la liste de gauche au sommet du module de code, sélectionner l'option UserForm. Dans la liste de droite au sommet du module de code, sélectionner l'événement Initialize.

3) Compléter la procédure événementielle UserForm_Initialize qui apparaît comme suit :

```
Private Sub UserForm_Initialize()
   Dim i As Integer
   i = 1
   Do Until IsEmpty(Worksheets("Divers").Range("A" & i))
      Call Me.ComboBoxTitre.AddItem(Worksheets("Divers"). _
      Range("A" & i).Value)
      i = i + 1
   Loop
End Sub
```

Remarque : **Pour écrire une instruction sur plusieurs lignes**, il faut, à la fin de chaque ligne, taper sur la barre d'espace et sur le trait continu (_).

4) Pour tester cette procédure, exécuter la procédure ProgPrincSaisieClients du module de code ModuleSaisieClients.

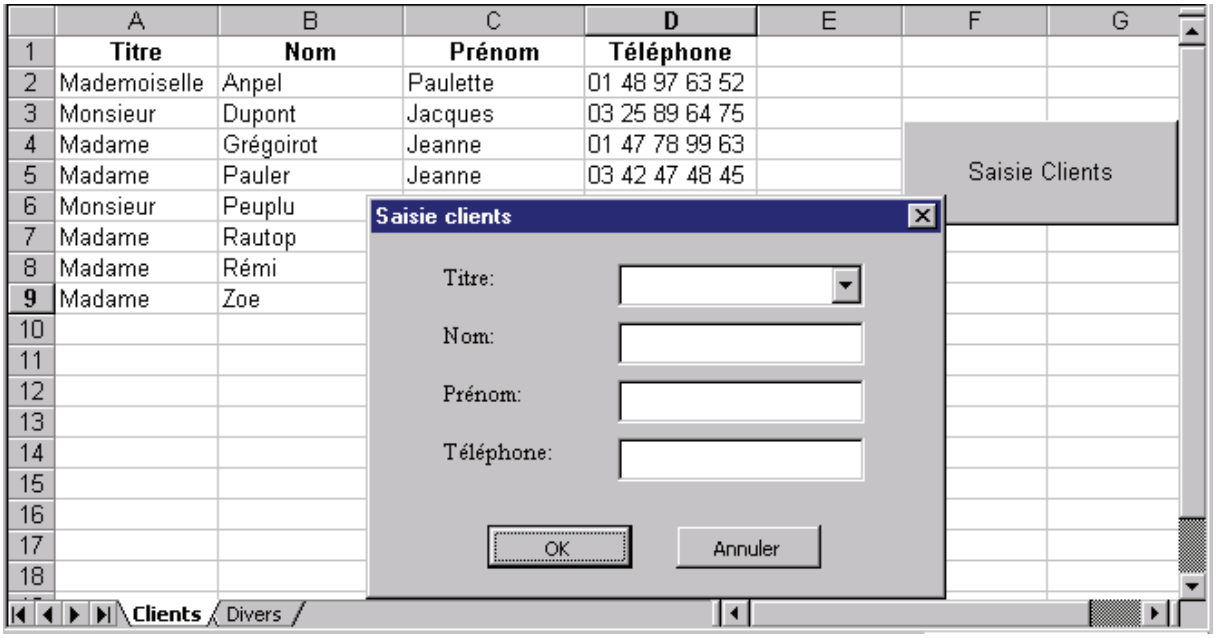
Pour information : Il existe une autre méthode pour initialiser la zone de liste modifiable ComboBoxTitre avec les valeurs "Madame", "Mademoiselle" et "Monsieur", en la liant à la plage de cellule A1:A3 de la feuillle de calcul DIVERS.

Pour lier un contrôle à une plage de cellules, il faut renseigner la propriété ROWSOURCE du contrôle avec la plage de cellules voulue (dans notre cas, Divers!A1:A3) ou encore mieux avec le nom de la plage de cellules voulue, ce qui suppose de nommer cette plage de cellules.
Remarque : La propriété CONTROLSOURCE permet de lier un contrôle à une seule cellule.

Attention: Cette méthode, contrairement à la précédente, ne pourra pas prendre en compte les éventuelles modifications faites sur le titre d'un client (ajout ou suppression d'un titre). La zone de liste modifiable ComboBoxTitre est liée à la plage de cellule fixe A1:A3 de la feuillle de calcul DIVERS.

Accéder aux contrôles d'un objet UserForm

Jusqu'à présent, nous avons créé un objet UserForm pour saisir de nouveaux clients, et, ouvert et fermé cet objet. Cependant, si l'utilisateur saisit des données dans l'objet UserForm, ces dernières ne sont pas recopiées dans liste de données EXCEL correspondante.



La mise à jour de la liste de données EXCEL correspondant aux données du client doit être faite lorsque l'utilisateur clique sur le bouton OK. Elle consiste à insérer une nouvelle ligne dans la liste de données et à recopier les valeurs des contrôles de l'objet UserForm dans les bonnes cellules de la nouvelle ligne.

1) Aller dans l'éditeur de Visual Basic. Dans l'explorateur de projets, selectionner l'objet UserForm de nom FenetreSaisieClients, ouvrir son module de code et compléter la procédure événementielle ButtonOK_Click comme suit :

```
Private Sub ButtonOK_Click()
   Sheets("Clients").Rows(2).Insert
   Range("A2:D2").Font.Bold = False
   Sheets("Clients").Range("A2").Value=Me.ComboBoxTitre.Text
   Sheets("Clients").Range("B2").Value = Me.TextBoxNom.Text
   Sheets("Clients").Range("C2").Value=Me.TextBoxPrénom.Text
   Sheets("Clients").Range("D2").Value = Me.TextBoxTél.Text
'tri de la liste de données dans l'ordre croissant des noms (deuxième colonne).
   Call Sheets("Clients").Range("A1"). _
   Sort(Key1:= Sheets ("Clients").Columns("B"), Header:=xlYes)
   Call Unload(Me)
End Sub
```

Attention : Si, dans les procédures événementielles ButtonOK_Click et ButtonAnnuler_Click, on utilise la méthode Hide pour masquer l'objet UserForm à la place de l'instruction Unload, les ressources mémoires de l'objet UserForm ne sont pas libérées. Lors d'un nouvel affichage de l'objet UserForm avec la méthode Show, les différents contrôles de l'objet ont alors les mêmes valeurs que lorsqu'il a été masqué.

VIII.4. AFFICHER UN OBJET USERFORM A PARTIR D'UN BOUTON D'UNE FEUILLE DE CALCUL

On veut pouvoir afficher la boîte de dialogue FenetreSaisieClients à partir d'un bouton de commande de la feuille de calcul CLIENTS.

1) Aller dans EXCEL et cliquer sur l'onglet CLIENTS. Positionner le curseur sur une cellule vide de la feuille de calcul CLIENTS et activer la commande AFFICHAGE BARRE D'OUTILS FORMULAIRES.

2) Une barre d'outils FORMULAIRES apparaît. Cliquer sur l'objet BOUTON, puis cliquer sur la feuille de calcul CLIENTS.

3) Dans la nouvelle fenêtre AFFECTER UNE MACRO qui apparaît, sélectionner la macro ProgPrincSaisieClients et cliquer sur le bouton OK.

4) Cliquer sur le bouton BOUTON 1 et renommer le SAISIE CLIENTS.

	A	B	C	D	E	F	G
1	**Titre**	**Nom**	**Prénom**	**Téléphone**			
2	Mademoiselle	Anpel	Paulette	01 48 97 63 52			
3	Monsieur	Dupont	Jacques	03 25 89 64 75			
4	Madame	Grégoirot	Jeanne	01 47 78 99 63		Saisie Clients	
5	Madame	Pauler	Jeanne	03 42 47 48 45			
6	Monsieur	Peuplu	Jean	02 58 96 69 58			
7	Madame	Rautop	Anelise	01 42 58 96 35			
8	Madame	Rémi	Yvette	01 25 78 41 32			
9	Madame	Zoe	Jeanne	02 45 78 91 25			
10							
11							
12							
13							

5) Fermer la barre d'outils FORMULAIRES et tester le bouton SAISIE CLIENTS.

VIII.5. EXERCICE

Ecrire un programme qui permet de convertir des devises.

- Vous définirez les devises qui vous intéressent dans la colonne A de la feuille de calcul DEVISES.

- Vous créerez un objet UserForm qui vous permettra de saisir la devise de départ à partir d'une zone de liste modifiable initialisée avec les devises définies précédemment, la devise d'arrivée à partir d'une zone de liste modifiable initialisée avec les devises définies précédemment et la valeur à convertir.
- Vous écrirez le code vous permettant d'effectuer la conversion et d'afficher la valeur résultat dans une boite de dialogue. Vous vous assurerez que la valeur à convertir est bien un nombre.

IX. BIBLIOGRAPHIE

Anne Caracache. *Excel 5 pour Windows, Formation Rapide, Perfectionnement*. Dunod, 1994.

Jean-François Sehan. *Excel 5 pour Windows, Macros Visual Basic, Formation Rapide, Perfectionnement*. Dunod, 1995.

Paul McFedries. *Excel pour Windows 95, Secrets d'experts*. SAM'S, 1995.

Mikaël Bidault. *Excel & VBA*. CampusPress, 2002.

Rob Bovey, Stephen Bullen, John Green, Robert Rosenberg. *VBA pour Excel 2002 La référence du programmeur*. CampusPress, 2002.