

AGROPARISTECH

PROGRAMMATION EN VBA(*) POUR EXCEL

U.F.R. D'INFORMATIQUE

Christine MARTIN

Année 2012-2013



Document inspiré du cours de Juliette Dibie-Barthelemy (Janvier 2008)

* Visual Basic pour Applications

Introduction

Le **VBA (Visual Basic pour Application)** est un langage de programmation permettant d'utiliser du code dont la syntaxe est proche du langage **Visual Basic** pour exécuter les nombreuses fonctionnalités notamment des applications Microsoft Office et par exemple de l'Application **EXCEL** que vous êtes amenés à utiliser fréquemment dans votre spécialité.

Créer des programmes en VBA permet notamment d'automatiser des tâches répétitives réalisées sous EXCEL grâce à la possibilité de manipuler les objets propres à Excel (Classeurs, feuilles, cellules, ...), de stocker de l'information, d'effectuer des instructions sous conditions et, éventuellement, à plusieurs reprises en fonction d'un ou plusieurs critères.

Le langage VBA permet également de contrôler la quasi-totalité de l'IHM¹ notamment d'Excel, ce qui inclut la possibilité de manipuler les fonctionnalités de l'interface utilisateur comme les menus, les barres d'outils et le fait de pouvoir personnaliser les boîtes de dialogue et les formulaires utilisateurs. Cela permet notamment de rendre une application développée sous EXCEL plus conviviale.

Il permet enfin de contrôler une application à partir d'une autre (par exemple, créer automatiquement un document WORD à partir de données sous EXCEL).

Ce document organisé en sections constitue une synthèse des possibilités offertes par la programmation en VBA sous Excel.

¹ Interface Homme-Machine : c'est-à-dire tous les médias qui permettent à un utilisateur d'interagir avec la machine

Table des matières

1	<i>Prise en main de l'environnement de développement</i>	5
1.1	Première macros ... utilisation de l'enregistreur de macros	5
1.1.1	Enregistrer une macro.....	5
1.1.2	Exécuter une macro	7
1.1.3	Supprimer une macro.....	8
1.2	L'environnement de développement Visual Basic Editor	8
1.2.1	Accès à l'environnement Visual Basic Editor (VBE)	8
1.2.2	Fenêtre de lecture, écriture et modification d'un programme.....	9
1.2.3	Affichage et modification des propriétés d'un module de code	11
1.2.4	Sauvegarde.....	12
1.2.5	Retrouver un programme enregistré : l'explorateur de projets	12
1.2.6	Exécuter directement dans VBE une procédure sans argument.....	13
1.2.7	Fermeture de l'environnement VBE.....	14
2	<i>Accès aux fonctionnalités d'Excel depuis VBA</i>	14
2.1	Les objets	14
2.2	Les collections	14
2.3	L'accès aux objets	14
2.4	Les propriétés des objets	15
2.5	Les méthodes des objets	16
2.5.1	Utiliser les fonctions intégrées d'Excel.....	16
2.6	Les événements	16
2.7	L'explorateur d'objets	17
2.7.1	Ouverture de l'explorateur d'objets	17
2.7.2	Utilisation de l'explorateur d'objets.....	17
2.8	Comment obtenir de l'aide : quelques astuces	19
3	<i>Vers la programmation avancée</i>	20

3.1	Stocker de l'information : les variables et les constantes	20
3.2	Définir ses propres fonctions	23
3.2.1	Définition.....	23
3.2.2	Utilisation sous Excel	24
3.2.3	Utilisation dans un programme VBA	25
3.3	Prendre en compte des conditions.....	25
3.4	Répéter les mêmes actions	26
3.5	Règles de bonne conduite en programmation	28
<i>Tutoriel : Les événements</i>		30
<i>Tutoriel : Les entrées et sorties standards</i>		32
<i>Tutoriel : Les Objets UserForm</i>		37

Table des figures

Figure 1 : Fenêtre de paramétrage et lancement de l'enregistrement d'une macro	5
Figure 2 : Fenêtre d'arrêt de l'enregistrement d'une macro (Excel 2003)	6
Figure 3 : Fenêtre d'exécution d'une macro	7
Figure 4 : Barre d'Outils Visual Basic (l'icône entourée permet de lancer VBE)	8
Figure 5 : Fenêtre principale de Visual Basic Editor (VBE)	9
Figure 6 : Fenêtre d'affichage d'un module de code.....	9
Figure 7 : Résultat de l'exécution de la macro Macro1	10
Figure 8 : Fenêtre de propriété d'un module de code	12
Figure 9 : Fenêtre "Explorateur de projets" dans VBE	12
Figure 10 : Fenêtre ouverte par clic droit sur l'explorateur de projets dans VBE	13
Figure 11 : Fenêtre de l'explorateur d'objets.....	17
Figure 12 : Résultat de la recherche de fonctions intégrées d'Excel.....	19
Figure 13 : Fenêtre d'erreur de compilation	21

1 Prise en main de l'environnement de développement

1.1 Premières macros ... utilisation de l'enregistreur de macros

Un programme écrit en VBA est souvent appelé, par abus de langage, une **macro**.

Dans un premier temps, une macro peut être créée en utilisant l'**enregistreur de macros**, qui ne nécessite aucune connaissance du langage VBA.

L'enregistreur de macro est un outil accessible notamment sous Excel et permettant d'écrire du code VBA à partir d'opérations effectuées manuellement dans l'application.

A l'image d'une caméra, toute la séquence d'actions effectuées entre le début et la fin de l'enregistrement est traduite en instructions VBA et stockée dans une procédure VBA sans arguments, qui pourra être exécutée à nouveau à l'identique.

Une procédure est un ensemble d'instructions VBA identifiés par un nom et ne renvoyant pas de valeur.

L'enregistreur de macro sous Excel (Outils / Macro / Nouvelle Macro) permet donc de générer facilement du code VBA et de découvrir les mots clés et syntaxes à employer pour écrire ensuite directement ses propres programmes.

1.1.1 Enregistrer une macro

1) Paramétrage de la macro

Activer la commande Outils Macro Nouvelle macro.

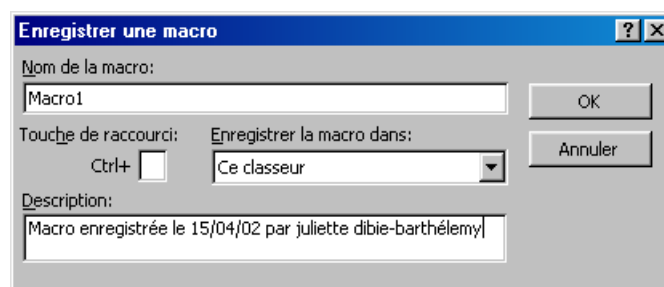


Figure 1 : Fenêtre de paramétrage et lancement de l'enregistrement d'une macro

La fenêtre présentée en

Figure 1 apparaît à l'écran et vous permet de paramétrer la macro qui va être créée c'est-à-dire :

- de préciser son nom dans le champ : **Nom de la macro**,
- de préciser à quel classeur elle sera rattachée par l'intermédiaire du menu déroulant **Enregistrer la macro dans**,
- d'y associer si besoin un raccourci clavier afin de faciliter son lancement futur : **Touche de raccourci**

- et enfin d'y associer une note explicative **Description**.

Le menu déroulant **Enregistrer la macro dans** vous propose trois options :

- Ce classeur,
- Nouveau classeur,
- Classeur de macros personnelles.

En sélectionnant, 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 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.

Enfin, l'option **Ce classeur** permet d'enregistrer la macro dans le classeur actif.

2) Lancement de l'enregistrement

Une fois votre macro paramétrée suivant vos besoins, Cliquer sur le bouton OK.

L'enregistrement est alors lancé et à partir de là tout les actions que vous faites dans Excel sont enregistrées, traduites en code VBA puis sauvegardées dans une procédure portant le nom que vous avez spécifié à l'étape précédente. Par exemple Macro1 comme indiqué dans la

Figure 1.

La barre d'outils ARRET DE L'ENREGISTREMENT apparaît, ce qui marque le début de l'enregistrement.



Figure 2 : Fenêtre d'arrêt de l'enregistrement d'une macro (Excel 2003)

Cette fenêtre permet également de déterminer le mode suivant lequel les cellules seront considérées dans le programme final.

Il existe en effet deux moyens de voir les choses :

- Soit on a une vision absolue : par exemple on considère la cellule A1 qui a une position déterminée dans une page de calcul Excel
- Soit on a une vision relative : on veut par exemple qu'une action soit faite toujours 2 cellules en dessous de celle sélectionnée à ce moment là.

Le bouton de droite de la fenêtre d'arrêt de l'enregistrement (Figure 2) permet de faire un choix entre ces deux modes. Ce choix dépend bien entendu des propriétés attendues de la macro que l'on est en train d'enregistrer.

Attention : Vérifier toujours le mode dans lequel vous vous trouvez au lancement de l'enregistrement et avant d'effectuer la moindre action sans quoi votre macro pourrait avoir un comportement différent de celui que vous attendiez car non enregistrée dans le mode adéquate.

Par ailleurs, 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**. Aussi et 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) Arrêt de l'enregistrement

Une fois que vous avez réalisé toutes les actions que vous souhaitez enregistrer pour les rejouer ultérieurement, l'arrêt de l'enregistrement se fait en cliquant sur le carré bleu de la barre d'outils **ARRET DE L'ENREGISTREMENT** (Figure 2).

Attention : n'oubliez pas d'arrêter l'enregistrement d'une macro ! Dans le cas contraire vos actions continueraient à être enregistrées et vous ne seriez pas en mesure de relancer cette macro.

L'enregistreur de macro permet de manière rapide et simple de créer des macros. Cependant une macro ainsi créée ne s'exécutera que sur un ensemble de cellules données et possèdera des fonctionnalités limitées. Pour pouvoir créer des macros propres à ses besoins, efficaces et interactives, la connaissance du langage VBA est nécessaire et il sera présenté dans les sections suivantes.

1.1.2 Exécuter une macro

Pour exécuter une macro déjà enregistrée, donc connue du système, il faut :

- 1) Activer la commande OUTILS MACRO/MACROS.

La fenêtre présentée en Figure 3 s'ouvre.

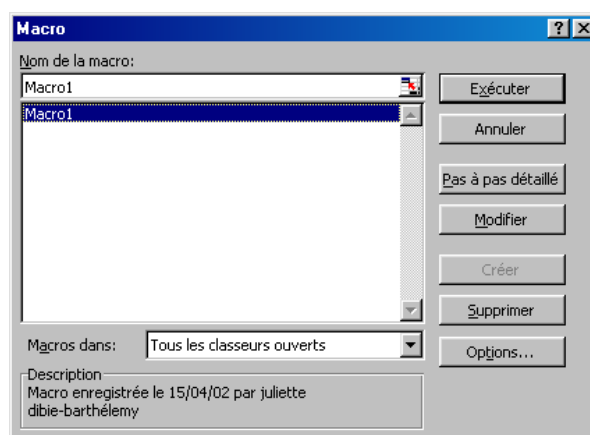


Figure 3 : Fenêtre d'exécution d'une macro

- 2) Sélectionner dans la liste la macro désirée et cliquer sur le bouton EXECUTER.

Dans l'exemple présenté en Figure 3, seule la macro nommée Macro1 est disponible.

Le menu déroulant **Macros dans** vous permet de faire un tri dans les macros disponibles et de n'afficher par exemple que celles enregistrées dans votre classeur personnel.

1.1.3 Supprimer une macro

Si vous devez supprimer une macro déjà créée, suivez les instructions suivantes :

- 1) Activer la commande Outils Macro macros.
La fenêtre présentée en Figure 3 s'ouvre.
- 2) Sélectionner dans la liste la macro désirée et cliquer sur le bouton SUPPRIMER.

1.2 L'environnement de développement Visual Basic Editor

Comme indiqué précédemment, l'enregistreur de macros permet de créer rapidement des programmes VBA simples. Pour aller plus loin, il faut connaître le langage VBA et apprendre à écrire soi-même les instructions qui correspondent aux actions que l'on souhaite faire réaliser par notre programme.

Pour cela on dispose sous Excel d'un environnement spécifique appelé Visual Basic Editor qui s'exécute dans une fenêtre différente de celle d'EXCEL.

Cet environnement permet de modifier des codes existants (produits par exemple par l'enregistreur de macros), de créer de nouveaux programmes « from scratch » (*ex nihilo*, à partir de rien), d'organiser ses programmes suivant leurs rôles dans des **modules**, de créer des **procédures** et **fonctions** (ensemble d'instructions identifiées par un nom unique et renvoyant une valeur, qui pourront être communes à différents programmes ou utilisables directement dans les feuilles de calcul, ... (cf. section 3.2)

1.2.1 Accès à l'environnement Visual Basic Editor (VBE)

L'environnement s'ouvre :

- Soit, en activant la commande : Outils / Macro / Visual Basic Editor
- Soit, en activant la commande AFFICHAGE BARRE D'OUTILS VISUAL BASIC et en cliquant sur l'objet VISUAL BASIC EDITOR (cf. Figure 4).

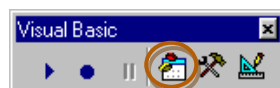


Figure 4 : Barre d'Outils Visual Basic (l'icône entourée permet de lancer VBE)

Dans les deux cas de figure, la fenêtre présentée en Figure 5 s'ouvre et vous donne accès à de nouvelles fonctionnalités.

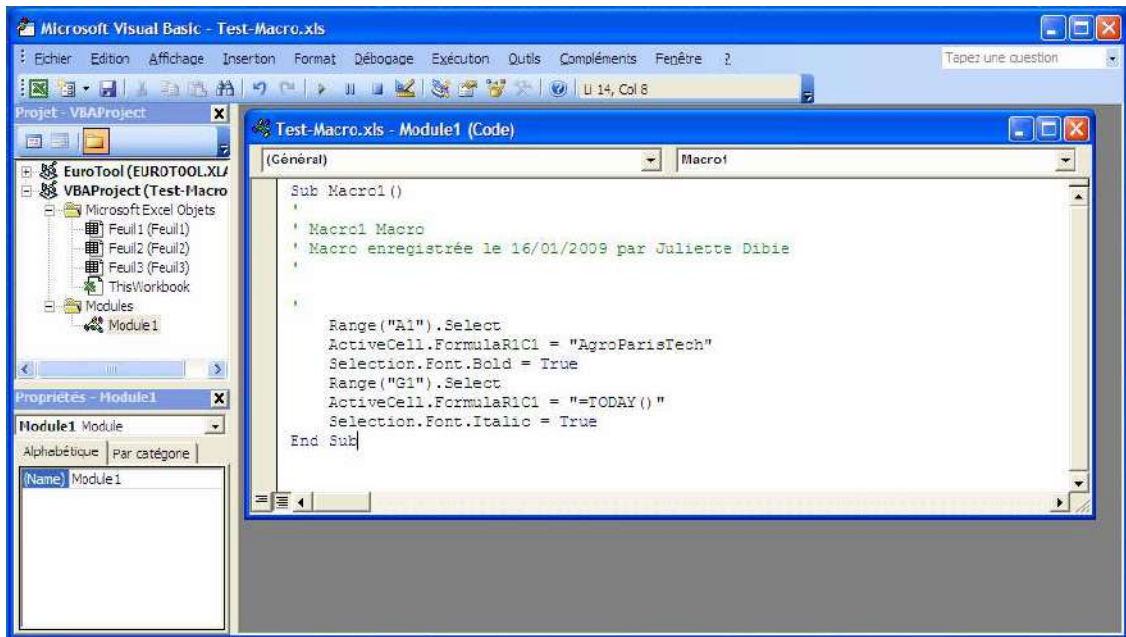


Figure 5 : Fenêtre principale de Visual Basic Editor (VBE)

Cette fenêtre se décompose notamment en trois parties :

- affichage, écriture et modification des fichiers de programmes disponibles ou modules de codes
- affichage des propriétés du module de code sélectionné
- explorateur de l'ensemble des modules de codes

Les particularités de ces zones sont détaillées dans les sections suivantes.

1.2.2 Fenêtre de lecture, écriture et modification d'un programme

Toutes les instructions VBA sont enregistrées dans un module de code qui est affiché dans la fenêtre principale de VBE à droite (cf. Figure 5 et Figure 6).

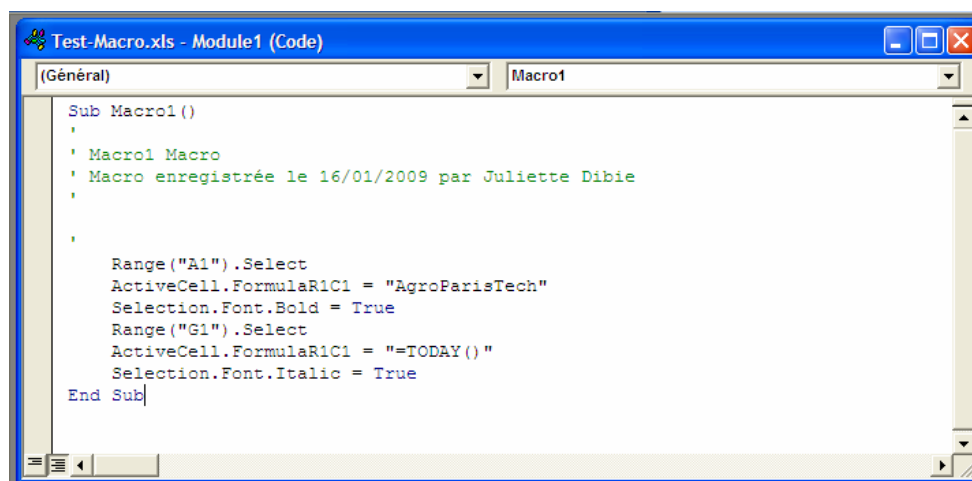


Figure 6 : Fenêtre d'affichage d'un module de code

Cette fenêtre se comporte comme un éditeur de texte classique avec quelques fonctionnalités supplémentaires telles que la coloration du texte en fonction de la nature de ce qui est écrit :

- Les mots clés du langage sont affichés en bleu
 - par exemple : **Sub** (mot clé de début de procédure), **True** (constante booléenne), ...
- Les commentaires (texte non interprété, utile pour comprendre ce que fait le programme) sont affichés en vert
 - Par exemple : « Macro enregistrée le »

Un programme est une suite d'instructions. Une **instruction** exécute une tâche précise. Elle est généralement écrite sur une seule ligne.

Exemple :

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

Cette instruction permet de sélectionner la cellule nommée A1 dans le classeur courant.

L'enregistreur de macro ne génère que des procédures sans arguments. 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 :

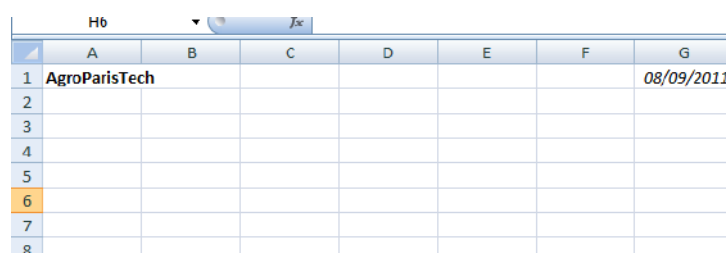
```
Sub NomProcédure([argument_1,..., argument_n])
    Instructions
    ...
End Sub
```

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

Par exemple, dans la Figure 6, la fenêtre d'affichage de modules de codes affiche le module nommé Module1 (cf. bandeau de la fenêtre). Ce module contient une seule macro nommée Macro1 (terme suivant le mot clé **Sub**).

Dans cette macro, il y a 5 lignes de commentaires (commençant par un ') puis 6 instructions. 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).

Lorsque cette macro est lancée on observe donc dans Excel le résultat suivant :



	A	B	C	D	E	F	G
1	AgroParisTech						08/09/2011
2							
3							
4							
5							
6							
7							
8							

Figure 7 : Résultat de l'exécution de la macro Macro1

Les macros produites par l'intermédiaire de l'enregistreur de macro sont également stockées dans des modules de codes.

Un module de code peut contenir plusieurs macros ou fonctions.

Créer une procédure directement dans VBE

Compte tenu de la syntaxe fournie ci-dessus, pour créer une nouvelle procédure directement dans VBE (sans passer par l'enregistreur de macros), il suffit de taper dans l'éditeur de texte de VBE, dans un module de code et en dehors de toute autre procédure, le mot clé Sub suivi du nom que vous voulez attribuer à votre macro, par exemple PrintValue, puis des parenthèses avec à l'intérieur les paramètres associés à cette procédure. Lorsque vous passez à la ligne, le mot clé End Sub de fin de procédure est automatiquement rajouté. Vous n'avez alors plus qu'à la compléter en fonctions de la tâche qu'elle doit remplir.

Utiliser une procédure dans une autre

Pour utiliser une procédure dans une autre, il faut utiliser le mot clé Call suivi du nom de la procédure que l'on souhaite utiliser.

Par exemple, ci-dessous, la procédure PrintValue est utilisée dans la procédure TestPrintValue.

```
Sub PrintValue(Value As Double)
    Range("B1").Value = Value
End Sub
```

```
Sub TestPrintValue()
    Call PrintValue(1.22)
End Sub
```

Remarque : Seules les procédures sans arguments sont exécutables directement par les menus ou icônes. Toute procédure ayant des arguments devra donc être appelée dans une autre procédure pour pouvoir être exécutée. Dans l'exemple précédent et en l'absence d'autres procédures, la procédure PrintValue ne pourra être exécutée que par l'intermédiaire de la procédure TestPrintValue.

Supprimer une procédure directement dans VBE

Pour supprimer une procédure directement dans VBE, il vous suffit dans l'éditeur de sélectionner le texte correspondant depuis le mot clé **Sub** jusqu'au mot clé **End Sub** et de le supprimer.

1.2.3 Affichage et modification des propriétés d'un module de code

La fenêtre en bas à gauche de l'environnement VBE, permet d'afficher et de modifier les propriétés du module sélectionné.



Figure 8 : Fenêtre de propriété d'un module de code

Dans la Figure 8, on peut constater qu'un module est uniquement caractérisé par un nom que l'on peut modifier par simple saisie dans cas se situant à droite du mot clé (Name).

1.2.4 Sauvegarde

Pour que les programmes produits soient conservés, il faut les sauvegarder à l'aide de la commande : FICHIER/ENREGISTRER ou le raccourci clavier CTRL+S.

1.2.5 Retrouver un programme enregistré : l'explorateur de projets

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

L'**explorateur de projets** (partie en haut à gauche de la fenêtre VBE (cf. Figure 5) affiche une liste hiérarchisée des différents projets VBA associés aux classeurs EXCEL ouverts (cf. Figure 9).

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

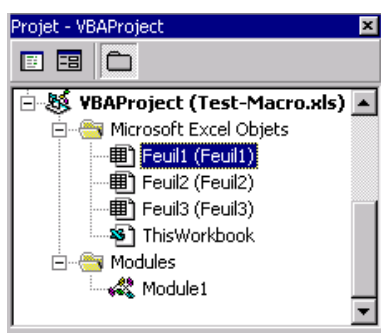


Figure 9 : Fenêtre "Explorateur de projets" dans VBE

Par exemple, le projet VBA associé au classeur TEST-MACRO.XLS visible en Figure 9 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.

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>).

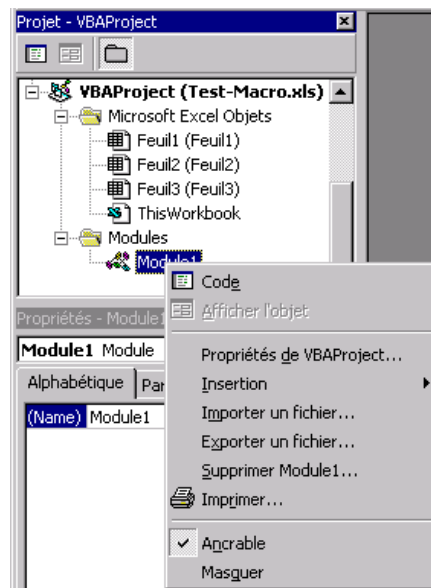


Figure 10 : Fenêtre ouverte par clic droit sur l'explorateur de projets dans VBE

L'ouverture d'un module de code peut également se faire par double-clic sur celui-ci dans l'explorateur de projets.


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


On peut insérer autant de modules de code qu'on le désire en fonction des besoins recensés. Dans la pratique, on cherche à regrouper dans un même module tous les éléments de programmes qui se rapportent à un ensemble de fonctionnalités donné. Par exemple, si l'on crée des fonctions de calcul mathématiques, on les regroupera dans un même module que l'on pourrait nommer Maths.

1.2.6 Exécuter directement dans VBE une procédure sans argument

Attention : 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 

1.2.7 Fermeture de l'environnement VBE

La commande **Fichier/Fermer et retourner à Microsoft Excel** permet de fermer VBE et de retourner dans Excel.

2 Accès aux fonctionnalités d'Excel depuis VBA

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. Comme nous le détaillerons dans les sections suivantes, dans EXCEL, les objets manipulés sont des classeurs, des feuilles de calcul, des plages, des cellules...

2.1 Les objets

VBA est un **langage de programmation orientée objet** attaché à une application. Tout est objet. En informatique, un objet est une entité nommée possédant des caractères propres (des propriétés) et actions propres (des méthodes).

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.

2.2 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.

2.3 L'accès aux objets

VBA permet de faire référence à un objet de différentes façons.

Par exemple, 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.
- Workbooks("Année2006").Sheets("Mars").Range("B2") désigne la cellule B2 de la feuille Mars du classeur Année2006
- Sheets("Mars").Range("B2") du classeur actif (ouvert et affiche) Range("B2") de la feuille active

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, en général être omis.

Les **références** aux cellules peuvent être :

- **absolues** : Range("B2") désigne la cellule B2 de la feuille active
- **relatives** à la cellule d'origine : ActiveCell.Offset (2,-1) désigne la cellule située 2 lignes plus bas (- pour plus haut) et une colonne à gauche (+ pour à droite) de la cellule active.

2.4 Les propriétés 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.

2.5 Les méthodes des objets

Les **méthodes** représentent les actions qui peuvent être effectuées par un objet.

Pour faire référence à une méthode d'un objet donné, il faut utiliser la syntaxe **Objet.Méthode** analogue à la syntaxe d'accès aux propriétés des objets.

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.

2.5.1 Utiliser les fonctions intégrées d'Excel

Toutes les fonctions intégrées d'Excel utilisable depuis VBA sont en fait des méthodes d'Objets particuliers pour VBA. Pour utiliser une fonction intégrée d'EXCEL dans VBA, il faut préciser à VBA où peut être trouvée cette fonction. Pour cela, il faut donc comme indiqué précédemment faire précéder le nom de la fonction de l'objet sur lequel elle va s'appliquer.

Par exemple, l'objet WorksheetFunction permet d'accéder à de nombreuses fonctions courantes telles que la fonction **Min**. Le code correspondant est donc :

```
WorksheetFunction.Min()
```

Attention : Toutes les fonctions intégrées d'EXCEL ne sont pas disponibles dans VBA ! Néanmoins, 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 terme de temps de calcul.

2.6 Les événements

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.

Pour qu'un objet réponde à un événement, il faut écrire du code VBA dans la procédure associée à l'événement considéré. Ces procédures sont stockées dans les modules de codes associés aux objets concernés.

Cf. Tutoriel : Gestion des évènements en VBA.

2.7 L'explorateur d'objets

Compte tenu du nombre important d'objets, propriétés, méthodes et événements disponibles pour Excel en VBA, on ne connaît par cœur dans la pratique qu'un petit sous ensemble. Il faut en revanche savoir retrouver les éléments dont on a besoin lorsqu'on développe un programme.

Pour cela, 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.

2.7.1 Ouverture de l'explorateur d'objets

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

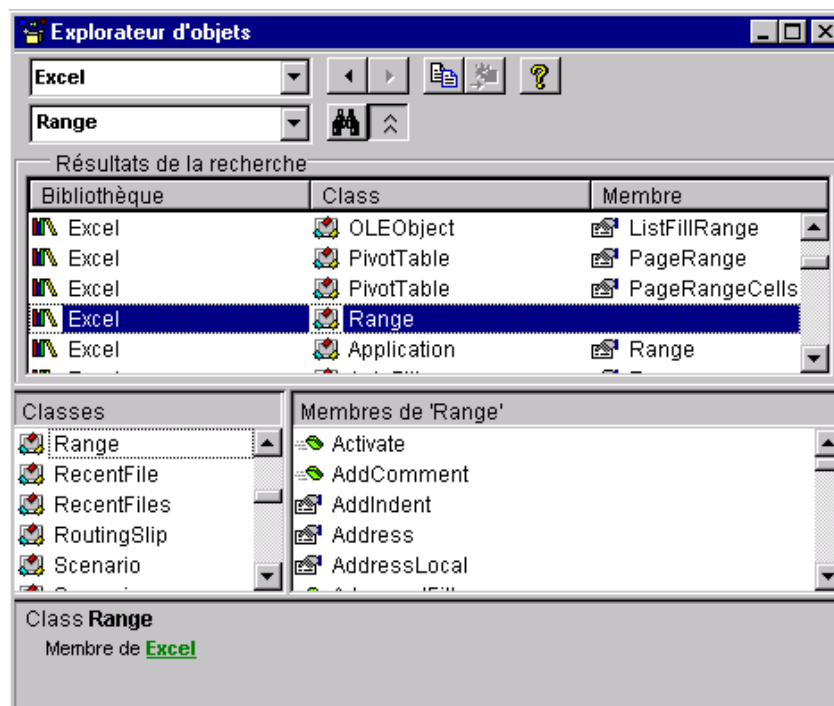


Figure 11 : Fenêtre de l'explorateur d'objets

La fenêtre présentée en Figure 11 : Fenêtre de l'explorateur d'objetsFigure 11 s'affiche.

2.7.2 Utilisation de l'explorateur d'objets

Sélection du champ de recherche

Le premier menu déroulant (cf. Figure 11) en haut à gauche (zone PROJET/BIBLIOTHEQUE) permet de sélectionner le projet ou la bibliothèque d'objets de son choix pour restreindre le champ des recherches. Vous pouvez par exemple ne sélectionner que la bibliothèque EXCEL afin de ne

rechercher que dans les fonctionnalités spécifiques à Excel et non dans celles communes à toutes les applications VBA par exemple.

Recherche par mots clés

La zone de saisie qui se situe juste en dessous (RECHERCHER TEXTE) permet de rechercher un élément (objet, propriété, fonction, évènement) dans l'espace défini dans le menu précédent. Taper l'élément recherché et appuyer sur la touche ENTREE. Le résultat de la recherche s'affiche dans la zone RESULTATS DE LA RECHERCHE qui se situe dans la partie basse de la fenêtre.





Affichage et lecture des résultats

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**.

Pour connaître, par exemple les fonctions intégrées d'EXCEL liées à l'objet Feuille de calcul et disponibles dans VBA (cf. section 2.5.1), activer la commande AFFICHAGE EXPLORATEUR D'OBJETS dans l'éditeur de Visual Basic et rechercher parmi les membres de l'objet `WorkSheetFunction`.

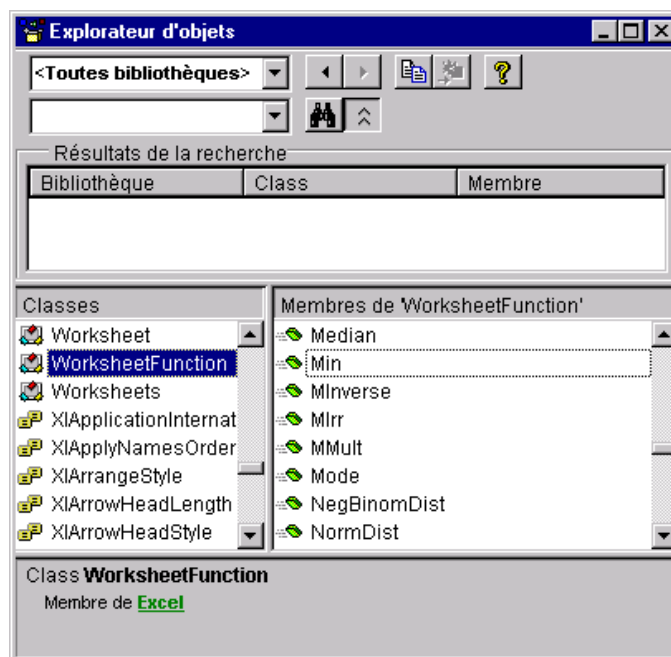


Figure 12 : Résultat de la recherche de fonctions intégrées d'Excel

La Figure 12 présente une vue des résultats proposé par l'explorateur d'objets pour cette recherche.

2.8 Comment obtenir de l'aide : quelques astuces

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

Une bonne méthode pour apprendre à se servir de VBA et découvrir de nouvelles instructions est de :

- tout d'abord enregistrer la séquence des actions dont on souhaite connaître les instructions correspondantes (cf. section 1.1.1)
- puis exécuter pas à pas (touche F8) dans l'outil VBA afin de savoir les actions effectuées par chacune des lignes de code et de s'en servir par exemple pour écrire, modifier ou compléter un programme.

- 2) Utiliser l'**explorateur d'objets** pour découvrir les objets disponibles dans VBA et les propriétés, méthodes et événements qui leur sont associés.
- 3) 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.

3 Vers la programmation avancée

Cette section a pour but de présenter ou rappeler suivant les cas les éléments du langage VBA qui sont communs à toutes les applications MICROSOFT OFFICE voire dans certains cas à tout langage de programmation.

3.1 Stocker de l'information : 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.

Syntaxe :

```
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.

En VBA la déclaration des variables n'est, par défaut, pas obligatoire. Néanmoins, il est préférable de forcer la déclaration explicite des variables notamment pour faciliter le contrôle du comportement du programme et traquer les erreurs de frappe très fréquentes en programmation.

Exemple :

```
Sub MacroTestVariable()  
    valeur = Range("A1").Value  
    Range("B1").Value = valleur  
End Sub
```

En effet, dans l'exemple précédent, quelle que soit la valeur de la cellule A1, cette macro n'écrira rien dans la cellule B1. Pourtant VBA ne détecte aucune erreur dans le code. Il considère qu'il y a deux variables distinctes **valeur** et **valleur**.

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 (en première ligne) des modules de code avant toutes procédures et toutes fonctions.

Si maintenant on reprend l'exemple précédent en forçant la déclaration des variables :

```
Option Explicit  
  
Sub MacroTestVariable()  
    valeur = Range("A1").Value  
    Range("B1").Value = valleur  
End Sub
```

Maintenant, VBA détecte une erreur à la compilation car la variable **valleur** n'est pas définie (cf. Figure 13).

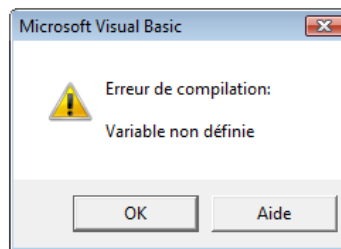


Figure 13 : Fenêtre d'erreur de compilation

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" '& : concatène les chaînes de caractères
```

```
Dim Age As Byte  
Age = 23
```

```
Dim MaPlage As Object `variable de type Object  
Set MaPlage = Range("A1:B5") `Set : pour affecter un objet
```

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).

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 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
```

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
```

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**.

3.2 Définir ses propres fonctions

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

3.2.1 Définition

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 donc la syntaxe suivante :

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



```
...  
End Function
```

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.

Une fonction indique la valeur à retourner en initialisant son nom avec la valeur de retour (cf. commentaire dans le cadre précédent).

Par exemple, une fonction fournissant la surface d'un cercle étant donné son rayon pourrait s'écrire comme suit :

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

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

Ici, **SurfaceCercle** est une fonction ayant 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.

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

3.2.2 Utilisation sous Excel

L'utilisation d'une fonction définie par l'utilisateur se fait dans Excel comme toute fonction intégrée.

Par exemple, pour la feuille de calcul suivante :

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

Si l'on note dans la cellule B6, la formule =SurfaceCercle(A6), et que l'on procède de manière analogue pour les cellules suivantes, on obtient le résultat suivant :

	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.

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).

3.2.3 Utilisation dans un programme VBA

Contrairement à une procédure, une fonction retourne une valeur qui lorsqu'on utilise cette fonction dans une autre fonction ou une autre procédure doit être sauvegardée dans une variable.

Par exemple, la fonction `SurfaceCercle` telle que définie en section 3.2.1 peut être utilisée dans la procédure `MacroTestFonction` dont le code est fourni ci-dessous.

```
Sub MacroTestFonction()  
    Dim valeur As Double  
    valeur = SurfaceCercle(1.5)  
    Range("B1").Value = valeur  
End Sub
```

Ici la variable **valeur** reçoit le résultat de la fonction **SurfaceCercle** pour un rayon donné égal à **1.5**.

3.3 Prendre en compte des conditions

Fréquemment dans un programme on souhaite que certaines tâches (instructions) soient réalisées dans certains cas et pas d'en d'autres. Ceci se fait en VBA par l'intermédiaire des mots clés **If**, **Then**, **Else** et **Elseif**.

Syntaxes :

<pre>If condition Then Instructions_si_vrai [Else Instructions_si_faux] End If</pre>	<pre>If condition_1 Then Instructions_1 ElseIf condition_2 Then Instructions_2 ElseIf condition_3 Then Instructions_3 ... Else Instructions_n End If</pre>
--	--

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 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
  Range("A1").Value = mes
End Sub

```

3.4 Répéter les mêmes actions

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

```

```

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

```

La boucle For...Next

La boucle **For...Next** est utilisée lorsque l'on connaît 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 = Range("A1").Value
    For compteur = 1 To n
        somme = somme + compteur
    Next compteur
    Range("A2").Value = "Somme des " & n & " premiers entiers : "
    Range("B2").Value = somme
End Sub
```

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

Exemple :

```
For compteur = 21 To 1 Step -2
    Range("A"&compteur).Value = compteur - 1
Next compteur
```

La boucle For Each...Next

La boucle **For Each...Next** permet d'accéder aux objets d'une collection.

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
```

Interrompre une boucle

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

Exemples :

- Exit Do, Exit For, Exit Function, Exit Sub,
- 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
```

La fonction InputBox permet de créer une boîte de dialogue. Pour plus d'informations à ce sujet reportez-vous au Tutoriel : Les entrées et sorties standards page 32.

3.5 Règles de bonne conduite en programmation

Afin que vos programmes soient lisibles et réutilisables, il est important de :

- Commenter vos codes
- Donner des noms explicites à vos variables, fonctions, procédures, ...
- Regrouper les ensembles d'instructions qui répondent à une fonctionnalité donnée soit :
 - dans une procédure : suites d'instructions identifiées par un nom unique et ne renvoyant pas de valeur ;
 - dans une fonction : une suite d'instructions identifiées par un nom unique et qui retourne une valeur.
- Réutiliser vos fonctions et procédures ou les fonctions intégrées d'Excel dans tout programme qui réclame les fonctionnalités correspondantes.
 - Autant que possible ne jamais dupliquer une même ligne de code !
- Regrouper vos fonctions et procédures en différents modules, nommés de manière explicite, suivant leurs finalités.

Excel-VBA et l'IHM

Sont présentées dans ce qui suit sous forme de tutoriels des fonctionnalités de VBA spécifiques à l'interfaçage entre l'utilisateur et Excel.

Tutoriel : Les événements **30**

Tutoriel : Les entrées et sorties standards **32**

Tutoriel : Les Objets UserForm **37**

Tutoriel : Les événements

Un **événement**, comme défini en section 2.6, 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.

L'objectif de ce tutoriel est d'écrire une procédure qui détecte chaque nouvelle plage de cellules ou cellule sélectionnée par l'utilisateur dans la feuille de calcul FEUIL1 du classeur nommé ici pour l'exemple TEST-MACRO.XLS et colorie son fond en bleu.

Avant d'écrire cette procédure, il faut :

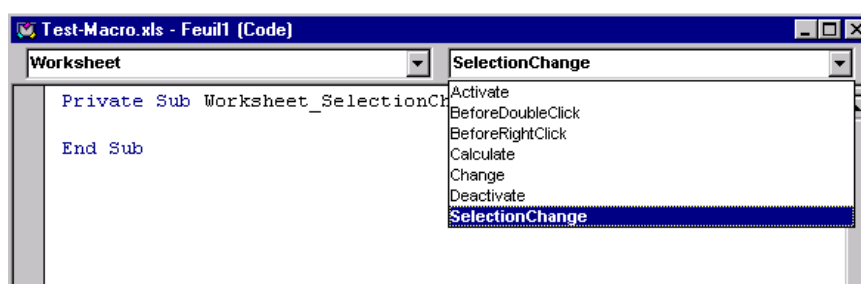
- identifier l'objet dont on veut traiter l'événement. Dans notre cas, il s'agit de la feuille de calcul FEUIL1 qui est un objet **Worksheet** ;
- puis identifier l'événement à traiter dans la liste des événements associés à l'objet considéré. Dans notre cas, on s'intéresse à chaque nouvelle sélection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

Pour cela, nous allons :

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, sélectionner 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).

Tutoriel : Les entrées et sorties standards

Dans ce tutoriel sont présentées deux fonctions d'entrées/sorties standards de VBA : la fonction MsgBox et la fonction InputBox.

1 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])
```

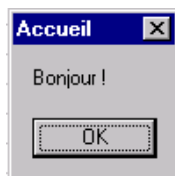
où prompt est le message affiché dans la boîte de dialogue, buttons correspond au type de boutons de la boîte de dialogue et title est le titre de la boîte de dialogue (cf. aide en ligne de l'éditeur de Visual Basic à l'aide de la touche F1.).

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 :

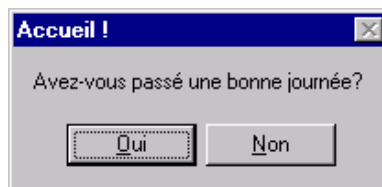
Dans l'instruction `Call MsgBox("Bonjour ! ", "Accueil")`, les valeurs des arguments sont spécifiées par position : les arguments doivent alors être saisis dans le bon ordre et des virgules supplémentaires doivent être incluses pour les arguments manquants.

Cette instruction peut être réécrite par nom en spécifiant les valeurs des arguments par nom comme suit : `Call MsgBox(prompt:="Bonjour ! ", Title:="Accueil")`. Dans ce cas, comme dans l'exemple précédent, il faut utiliser le signe `:=` entre le nom de l'argument et sa valeur.

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.

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` mais la syntaxe suivante :

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

où `reponse` est une variable dans laquelle on va stocker temporairement le résultat.

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

```

2 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"



3 Récapitulatif sur les parenthèses 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

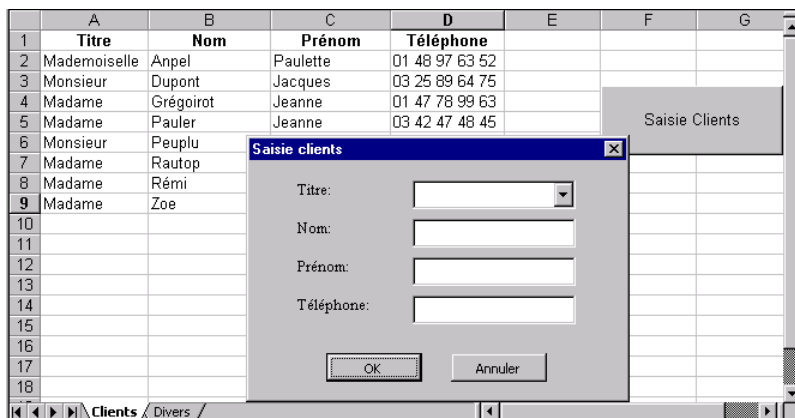
`Call MsgBox(prompt := "Bonjour !", Title := "Accueil")`

Tutoriel : Les Objets UserForm

Introduction

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

Ce tutoriel 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.

	A	B	C	D
1	Titre	Nom	Prénom	Téléphone
2				

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

	A
1	Madame
2	Mademoiselle
3	Monsieur

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

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

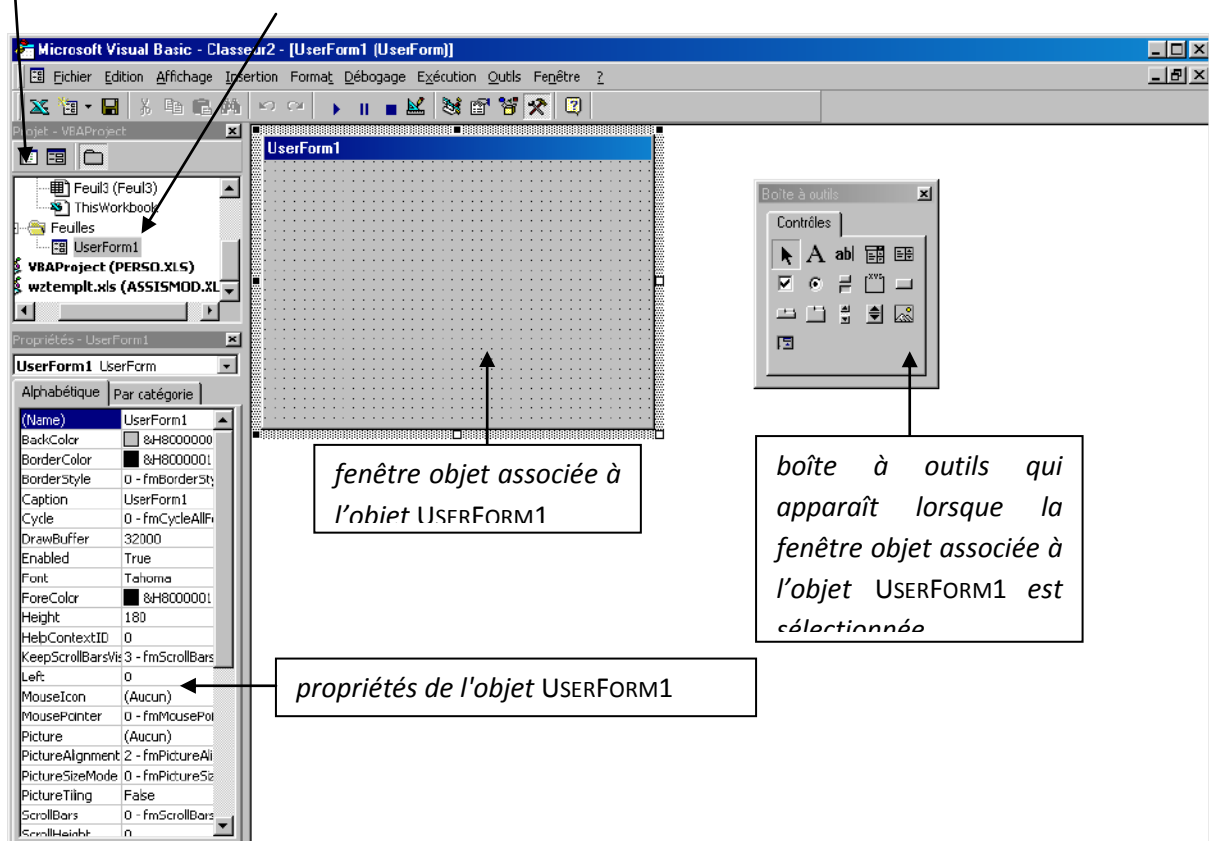
Il s'agit de déterminer le comportement de l'objet UserForm et de ses contrôles face aux différents événements utilisateur pouvant l'affecter et d'écrire le code permettant d'exploiter les actions effectuées par l'utilisateur.

1 Créer 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.

Dans l'explorateur de projets, les boîtes de dialogue sont regroupées dans un dossier FEUILLES

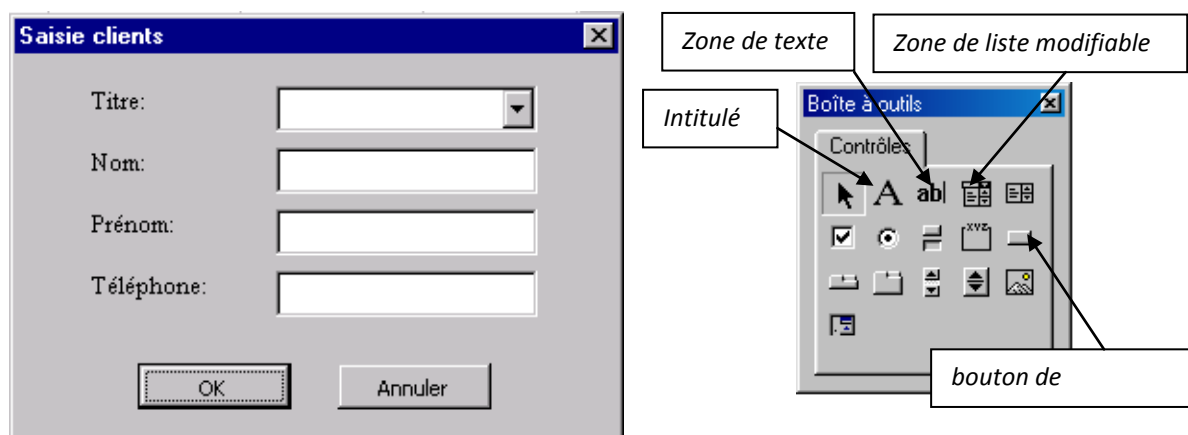
objet USERFORM1 qui est un objet de type FEUILLE et qui permet de dessiner une boîte de dialogue



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 INTITULÉ et une ZONE DE LISTE MODIFIABLE pour saisir le titre du client (Madame, Mademoiselle ou Monsieur), trois INTITULÉS 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
INTITULÉ	IntituléTitre	Titre :	police TIMES NEW ROMAN, taille 10
ZONE DE LISTE MODIFIABLE	ComboBoxTitre		"
INTITULÉ	IntituléNom	Nom :	"
ZONE DE TEXTE	TextBoxNom		"
INTITULÉ	IntituléPrénom	Prénom :	"
ZONE DE TEXTE	TextBoxPrénom		"
INTITULÉ	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.

2 Afficher et fermer un objet UserForm

2.1 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).

2.2 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.

3 Associer du code à 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 :

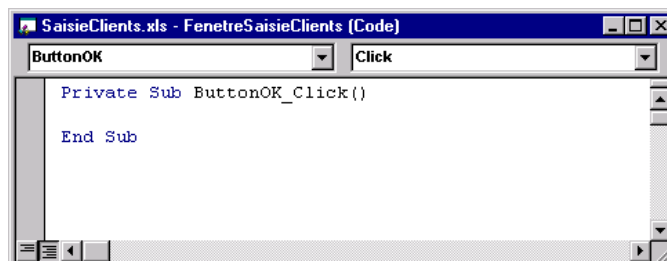
- Événement Change : détecté lors de la modification de la valeur (propriété VALUE) d'un contrôle (par exemple une zone de texte).
- Événement Click : détecté lorsque l'utilisateur clique sur un contrôle (par exemple un bouton de commande).
- Événement dblClick : détecté lorsque l'utilisateur double-clique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

3.1 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.

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

3.2 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.

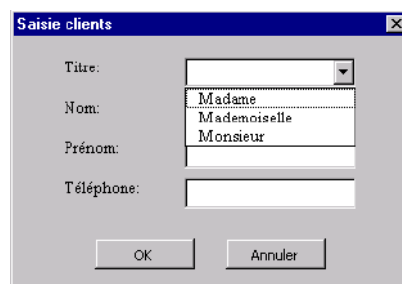
Attention : Lorsque la méthode Show est appliquée à un objet UserForm masqué par la méthode Hide, l'objet UserForm n'est pas rechargé mais uniquement affiché. L'événement Initialize n'est alors pas déclenché.

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 feuille de calcul DIVERS.



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.

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 feuille de calcul DIVERS.

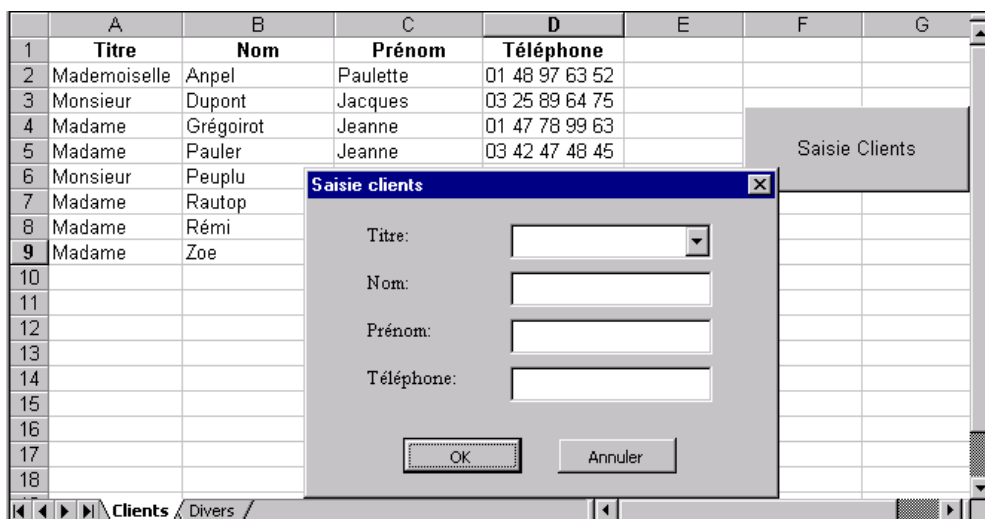
Pour lier un contrôle à une plage de cellules, il faut renseigner la propriété ROWSOURCE du contrôle avec la plage de cellules voulue (dans notre cas, Divers!A1:A3) ou encore mieux avec le nom de la plage de cellules voulue, ce qui suppose de nommer cette plage de cellules.

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 feuille de calcul DIVERS.

3.3 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, sélectionner 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
End Sub
```

```
'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.

4 Afficher un objet UserForm à 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`.

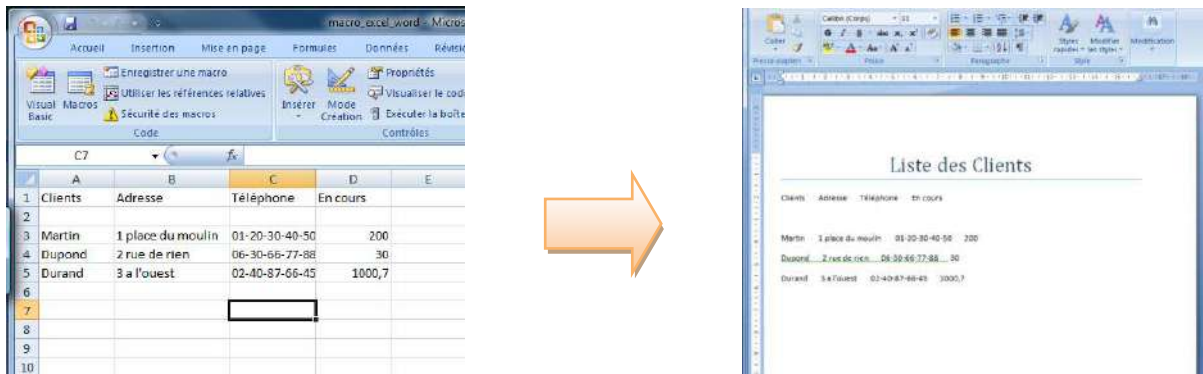
	A	B	C	D	E	F	G
1	Titre	Nom	Prénom	Téléphone			
2	Mademoiselle	Anpel	Faulette	01 48 97 63 52			
3	Monsieur	Dupont	Jacques	03 25 09 64 75			
4	Madame	Grégoire	Jeanne	01 47 78 99 63			
5	Madame	Fauler	Jeanne	03 42 47 48 45			
6	Monsieur	Peuplu	Jean	02 58 96 69 58			
7	Madame	Rautop	Anelise	01 42 58 96 35			
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`.

Ponts entre applications

Il est possible avec VBA de programmer des fonctionnalités avancées pour l'ensemble des applications Office et de ce fait de créer des « ponts » entre ces différentes applications. Par exemple, on peut retranscrire automatiquement dans un document Word le contenu d'un fichier Excel.



Pour cela dans une Macro Excel, il vous faudra :

- accéder à l'application Word : `CreateObject ("Word.Application")`
- rendre la fenêtre visible : `ObjetWord.Visible = True`
- ouvrir un document : `ObjetWord.Documents.Open`
- ajouter un document : `ObjetWord.Documents.Add`
- sauvegarder le document : `ObjetWord.Documents.Save`
- écrire du texte : `ObjetWord.Selection.TypeText Text:="le texte"`
- ...

Le code suivant est fourni en exemple :

```
Sub CopieWord()  
    Dim AppWord As Object  
    Set AppWord = CreateObject("Word.Application")  
  
    With AppWord  
        .Documents.Add  
        .Selection.TypeText Text:="Liste des Clients"  
        .Selection.TypeParagraph `sauter une ligne  
        `on copie le contenu de A1 dans le document Word  
        .Selection.TypeText Text:="" & Range("A1").Value  
        .Documents.Save  
        .Visible = True  
    End With  
End Sub
```

L'enregistreur de macro existe également sous Word et vous permettra de découvrir les instructions utiles à vos besoins.

Actions par URL

Dans un programme VBA on peut accéder à des outils extérieurs tels que des requêtes internet par l'intermédiaire de la fonction suivante :

- `ActiveWorkbook.FollowHyperlink(Url)`

« **Url** » représente ici des données concernant le service auquel on souhaite se référer. Cette fonction envoie alors une requête à travers le navigateur par défaut.

Interrogation d'un service web

Url du type : http://nom_du_service?requete-eventuelle

Si l'on considère par exemple l'Url : <http://www.google.com/search?q=ege+agroparistech> on pourra grâce à la fonction précédente lancer depuis une macro VBA dans Excel une recherche sur internet par le service Google pour les mots clés « ege » et « agroparistech ».

Tout site internet est accessible par ce biais. Le format des requêtes dépend de chaque service notamment au niveau des mots clés utilisés (pour google « q » représente la recherche formulée).

Envoie de mails

Grâce à l'instruction fournie précédemment, on peut également déclencher l'envoi d'un mail par l'intermédiaire d'une url du type :

« `mailto:adresse_mail?subject=le_sujet&body=le_contenu_du_message` »

Par exemple, pour l'url suivante mailto:edgar.poe@gmail.com?subject=Ohé&body=ca_va_?, edgar.poe@gmail.com représente l'adresse mail à laquelle on souhaite envoyer le message, « Ohé » représente le titre du message et « ca va ? » son contenu.