EXCEL 97-2000 VBA (us)
COURS

Michel Cabaré
Novembre 1999

TABLE DES MATIÈRES

LES VERSIONS DE VBA

Un nouveau langage :

C'est un langage issu du VISUAL BASIC, existant en tant que langage de développement de façon complètement indépendante

Apparu avec la version 5.0 d'EXCEL, le VBA est spécifiquement orienté application dans le sens où il intègre toute une série d'objets de propriétés et de méthodes spécifique à l'application sur laquelle il s'appui, ainsi qu'un véritable langage de programmation de type structuré

Voilà un petit récapitulatif des versions de VBA pour EXCEL

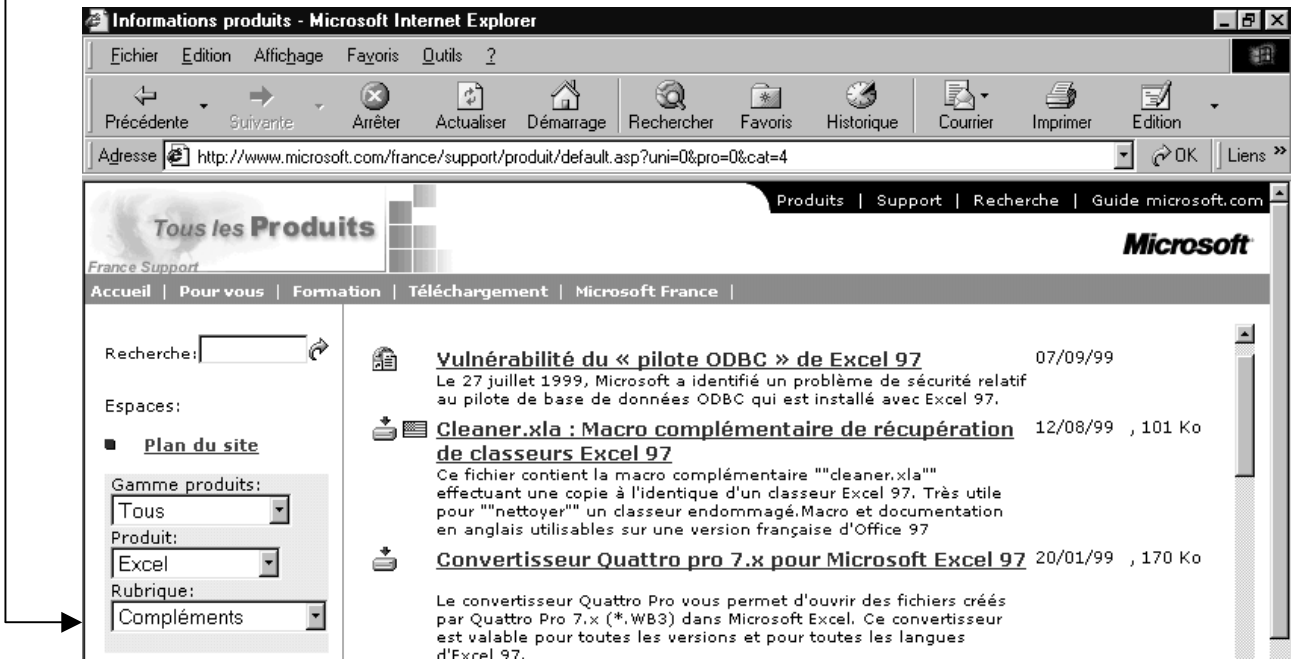
version Excel	version macro / VBA	caractéristique
Excel 4.0	macro 4.0	propriétaires à Excel
Excel 5.0	• macro 4.0 encore fonctionnelles • vba version 4.0 en Fr /Us	• en écriture / lecture • nouveau langage de type structuré
Excel 95 (ver 7.0)	• macro 4.0 encore fonctionnelles • vba version 4.0 en Fr /Us	• en écriture / lecture • quelques modifs mineures
Excel 97 (ver 8.0)	• macro 4.0 supportées • vba version 5.0 en Us uniquement	• en lecture / enregist seulement • avec ajout de fonctionnalités
Excel 2000 (ver 9.0)	• vba version 6.0 en Us	• avec ajout de fonctionnalités

CONVERSIONS DE MACRO ANCIENNES

Récupération d'information :

Un certain nombre de fichiers et d'utilitaires sont disponibles pour nous faciliter un peut la vie :

Fondamentalement on pourra récupérer ces fichiers sur Internet, sur le site microsoft dans l'espace **Complément**



d'autres utilitaires peuvent se trouver sur le site **www.baarns.com**

Macro Excel 4.0 en VB :

Aucun traducteur n'existe, car VB est censé avoir amené une programmation structurée, qui ne peut donc être générée à partir d'un code séquentiel écris en langage de macro-commandes Excel 4.0

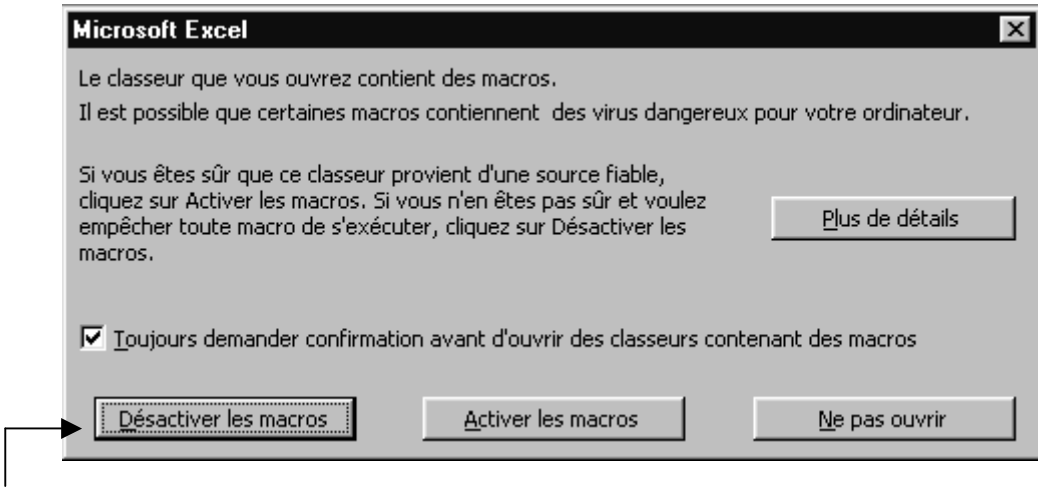
Avec Excel 95: vb fr en vb us :

Afin de vous faciliter la traduction en Anglais de vos applications réalisées en Visual Basic Edition Applications de langue française, un outil sous forme de macro complémentaire est disponible dans le dossier **TradVBA** du CD contenant Office95, voire sur le site internet de microsoft

1. Copier les 4 fichiers ci-dessous dans un dossier sur votre disque dur :
   US_FR.XLA Macro-complémentaire
   US_en_FR.XLT Modèle de module français
   FR_en_US.XLT Modèle de module anglais
   US_FR.HLP Fichier d'aide
2. Lancer Microsoft Excel
3. Ouvrer le fichier **US_FR.XLA**
   Une barre d'outils se crée vous permettant ensuite de traduire vos modules affichés.

Avec Excel 97 vb fr en vb us :

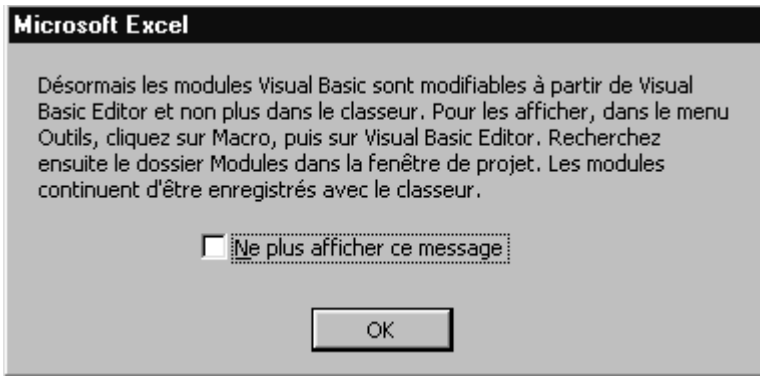
Lorsque l'on ouvre une macro avec excel 97 on obtient



Si vous **désactivez** les macros, vous ne pourrez pas les exécuter, ou recalculer des fonctions personnalisées…mais vous avez toujours la possibilité d'afficher et de modifier le texte des macros.

Lorsque vous cliquez dans le menu Fichier sur Enregistrer, les macros sont enregistrées. Vous pouvez fermer le classeur et l'ouvrir à nouveau en activant les macros si vous souhaitez les utiliser

Que l'on active ou pas les macros, une traduction automatique est effectuée par Excel97



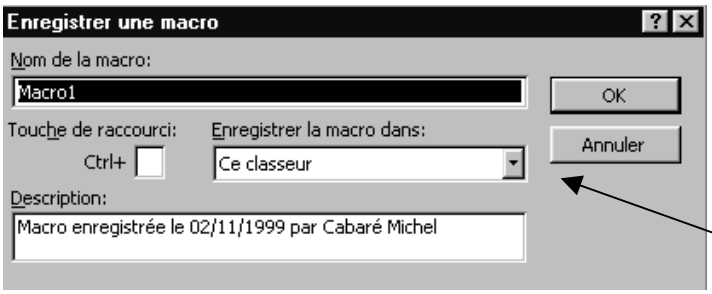
pour garder ces changements il faut bien sûr ensuite **enregistrer sous** en demandant explicitement un format de fichier Excel 97

MACRO ENREGISTRÉE

Démarrer l'enregistrement :

Pour enregistrer une macro on demandera simplement le menu

Outils / Macro / Nouvelle macro



Il est évident que la macro ne pourra pas toujours être stockées par défaut dans le classeur courant, mais pour l'instant cela convient

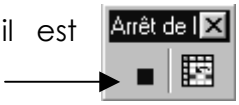
A partir de cet instant Excel mémorise toutes vos actions jusqu'à ce que vous stoppiez l'enregistrement:

Arrêter l'enregistrement :

Tant que l'inscription **Enregistrement** paraît dans la barre d'état, Excel mémorise toutes nos actions

Cet enregistrement peut s'arrêter de deux façons :

- soit par l'outils « arrêter l'enregistrement » s'il est disponible

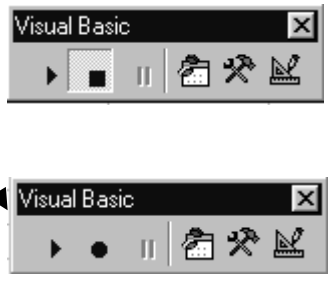


- soit par le menu **Outils / Macro / Arrêter l'enregistrement**

N.B: on peut à la rigueur faire apparaître dans la barre d'outils « visual basic » en demandant le menu

Affichage / Barre d'outils / Visual Basic

Bouton arrêt / démarrer un enregistrement



APPEL D'UNE MACRO

Menu Standard :

Il y a plusieurs façons d'exécuter une macro, cela dépends de la manière dont les options ont été définies lors de son enregistrement/création, ainsi que du contexte au moment de la demande d'exécution…

Si la macro existe, et que l'on veut l'exécuter il suffit de demander le menu

Outils / Macro / Macros… voire la combinaison **ALT** + **F8**

1) Choisir la macro voulue

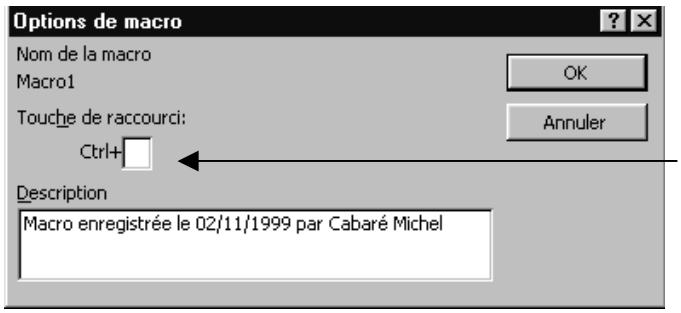
2) Exécuter

N.B: Modifier : change les actions écrites en vba

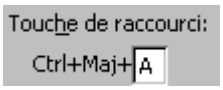
N.B: Options : change les paramètres d'appels de la macro

Combinaison de touches :

Cette méthode n'est disponible que lorsque dans les options d'enregistrement on a demandé **Touche de raccourci**



Si on tape une lettre le raccourci sera **CTRL+lettre** mais si on tape **MAJ+lettre** le raccourci sera **CTRL+MAJ+lettre**



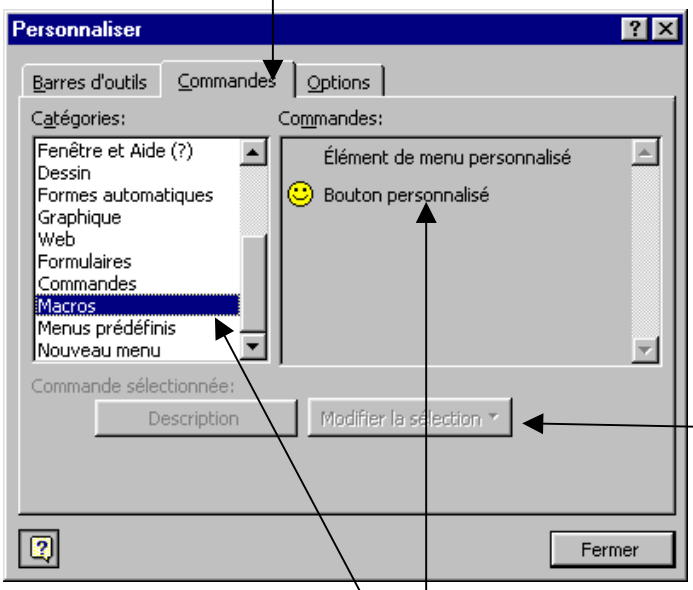
Outils Standard :

à partir de la barre d'outils visual basic demander "**Exécuter la Macro** " et sélectionner la macro voulue

Personnaliser une Barre d'outils standard :

On peut affecter la macro à un outils spécifique d'une barre d'outils:

- Cliquez avec le bouton droit sur une barre d'outils
- Cliquez sur **Personnaliser** dans le menu contextuel obtenu
- Choisissez l'onglet **Commandes**



- Cliquez sur Macros dans la liste des **Catégories**
- Faites glisser le **bouton personnalisé** sur la barre d'outils que l'on souhaite personnaliser.

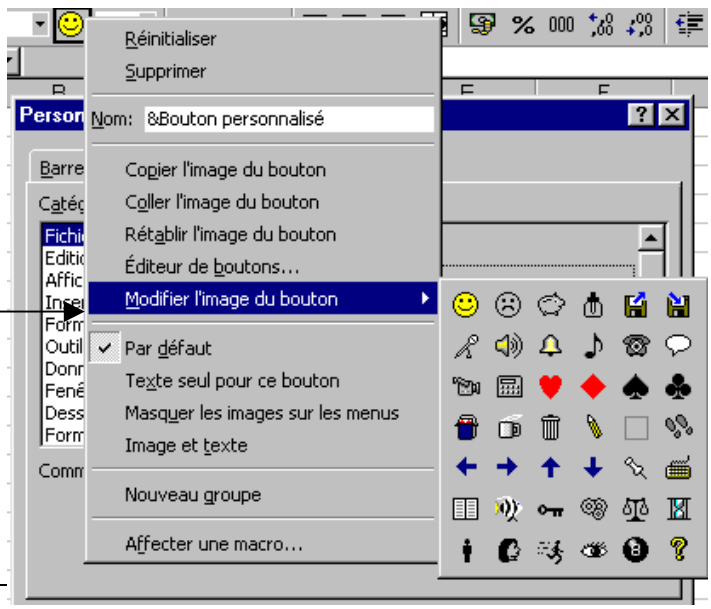
- Cliquez ensuite sur le bouton **Modifier la sélection** et choisissez Affecter une macro

- Venez double cliquer sur la macro concernée. Ne fermez pas ! ! !

Modifier l'apparence du bouton personnalisé

Vous êtes toujours dans la fenêtre **Personnaliser**

- Venez cliquez personnalisé (dans la barre d'outils)par le menu contextuel sur le bouton
- Choisissez Modifier l'image du bouton
- Choisissez le bouton désiré
- Fermer

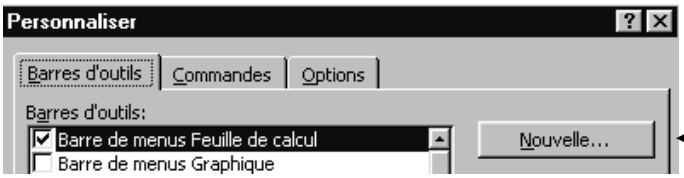


Barre d'outils personnelle :

On peut affecter la macro à un outils spécifique d'une barre d'outils. La procédure est la suivante:

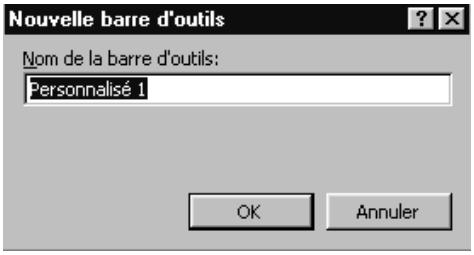
Création de la barre d'outils

Menu **Affichage/Barre Outils**,



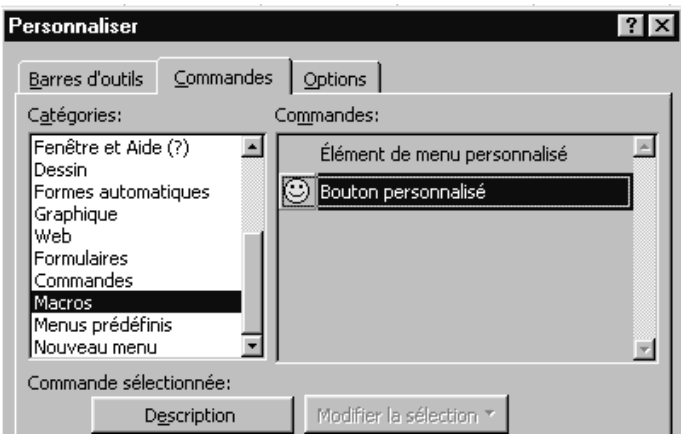
demander **Nouvelle…**

et lui donner un nom



Ajouter un outils

Pour ajouter un outils à cette barre d'outils il faut dans la boite de dialogue **personnaliser**, prendre l'onglet commande et choisir un bouton (de préférence n'ayant pas déjà une signification)

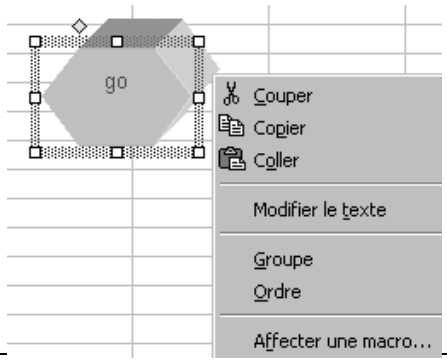


Puis on lui affectera une macro par le menu contextuel

Dessin dans la feuille :

N'importe quel dessin construit dans une feuille Excel peut être associé à une macro par un menu contextuel

Affecter une macro…



Commande personnalisée :

Il est possible d'ajouter une commande personnalisée aux menu d'Excel pour exécuter notre macro, et plus précisément de se créer des menus complets et hiérarchiques

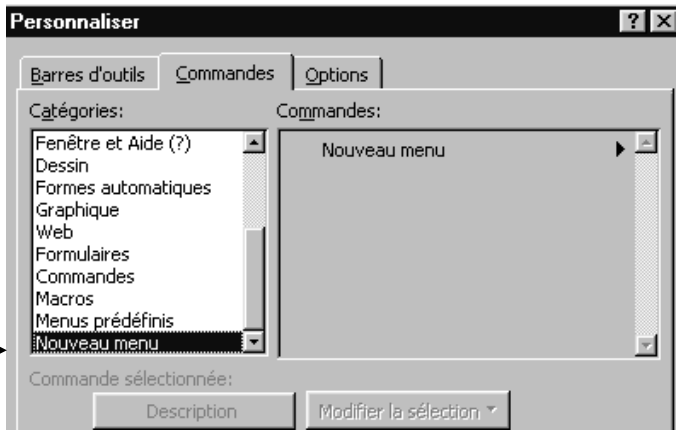
N.B : Ces commandes sont attachées à la feuille de macro uniquement

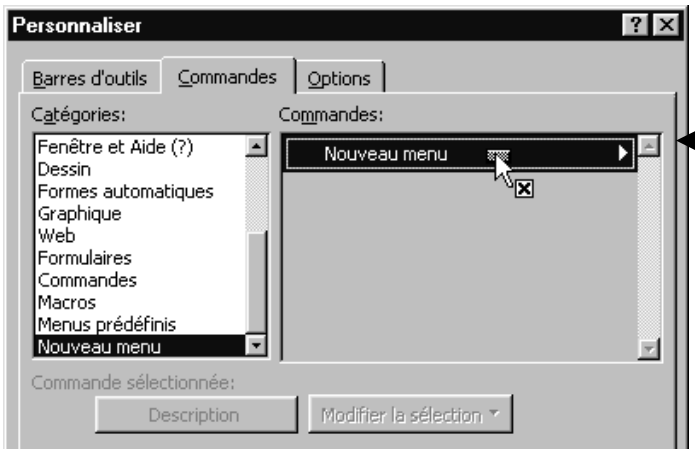
Création du menu

Il faut demander le menu

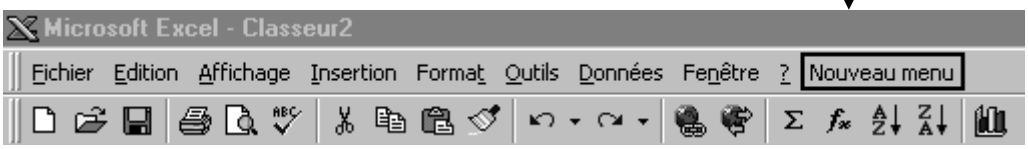
Outils / Personnaliser… et demander l'onglet **Commandes**

1) faire défiler jusqu'à Nouveau menu et sélectionner **Nouveau menu**



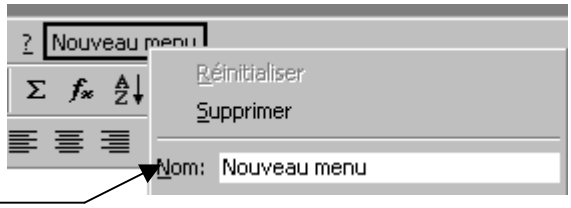


2) sélectionner **Nouveau menu** sur la droite et le faire glisser à l'emplacement souhaité dans la barre de menu



Avec un clic droit sur ce nouveau menu, donner un nom plus parlant dans la zone de texte prévue

comme "*Automatisation*"



Création des sous menu

Ayant toujours à l'écran la commande

Outils / Personnaliser… et dans l'onglet **Commandes**

On reproduit exactement la même chose mais en faisant glisser cette fois la souris **dans le menu personnalisé** de manière à créer un sous menu

et ainsi de suite

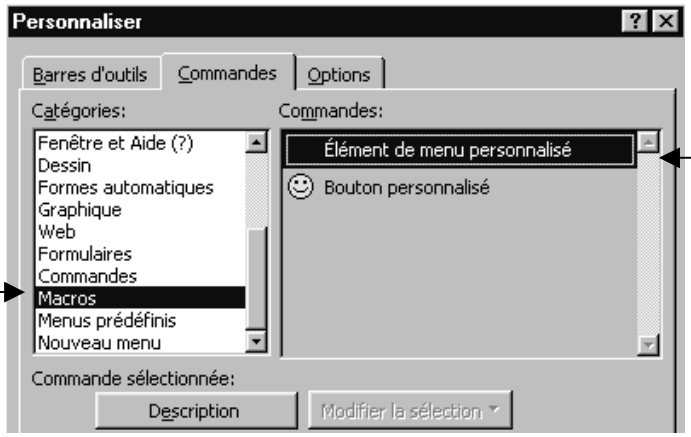
pour supprimer un menu ou sous menu il suffit de demander dans le menu contextuel **Supprimer**

Création des commandes textes

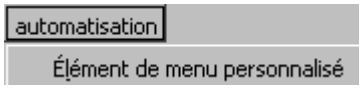
Ayant toujours à l'écran la commande

Outils / Personnaliser… et dans l'onglet **Commandes**

1) faire défiler et sélectionner **Macros**

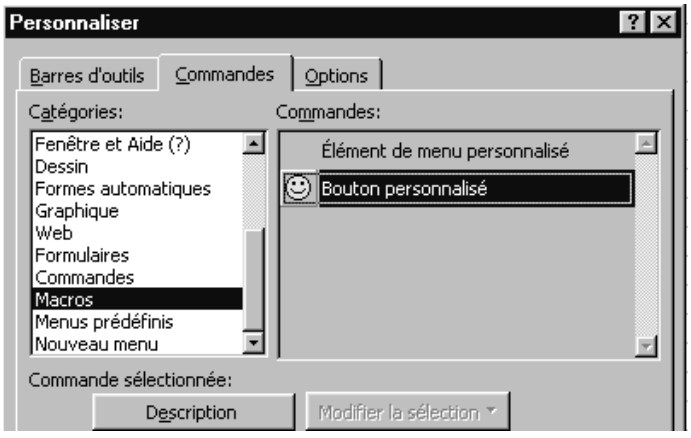


2) sélectionner **Elément de menu personnalisé** sur la droite et le faire glisser à l'emplacement souhaité dans le menu

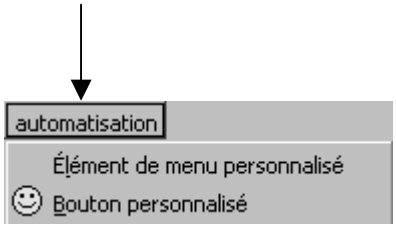


Création des commandes boutons

Idem ci-dessus mais en sélectionnant partie droite **Bouton personnalisé**



2) sélectionner **Boutons personnalisé** sur la droite et le faire glisser à l'emplacement souhaité dans le menu



N.B: un éditeur de bouton est disponible via le menu contextuel

PORTÉE D'UNE MACRO

Localisation de la feuille module :

Une macro c'est un langage de programmation stocké dans des feuilles particulières des classeurs excel , n'ayant plus rien à voir au niveau aspect avec une feuille classique. Ces feuilles sont nommées modules et sont visualisables uniquement à travers l'éditeur visual basic

A la différence de toutes les versions précédentes, la feuille contenant le code des macros fait toujours partie du classeur, mais n'est plus affichée en clair dans le classeur.

On peut visualiser ces feuilles en demandant le menu

Outils / macro / macros

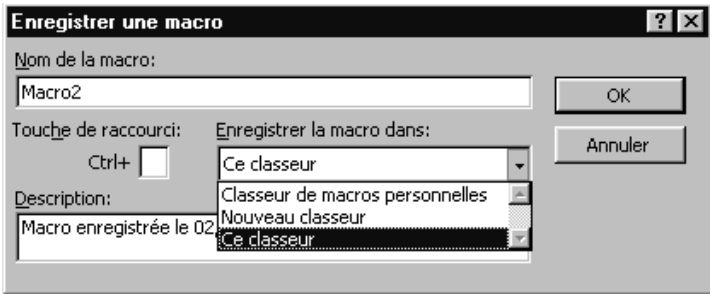
puis après avoir sélectionnée la macro voulue on demande **Modifier**. On se retrouve dans l'environnement de l'éditeur visual basic

On peut également aller dans l'éditeur visual basic en demandant le menu

Outils / macro / visual basic editor voire la combinaison **ALT** + **F11**

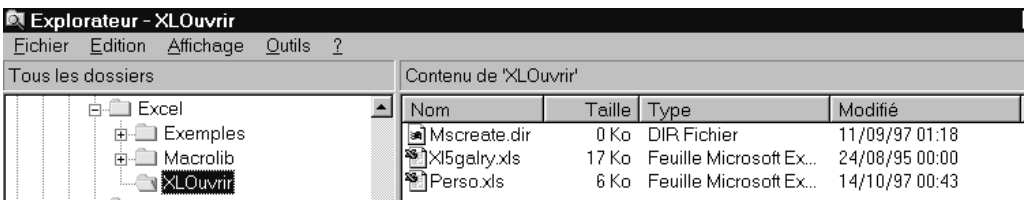
On se retrouve dans l'environnement de l'éditeur visual basic

Lors de la création d'une macro, Excel demande dans quel classeur on veut la stocker.

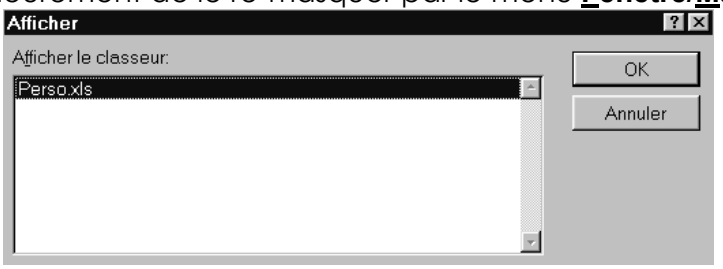


Classeur de macros personnelles :

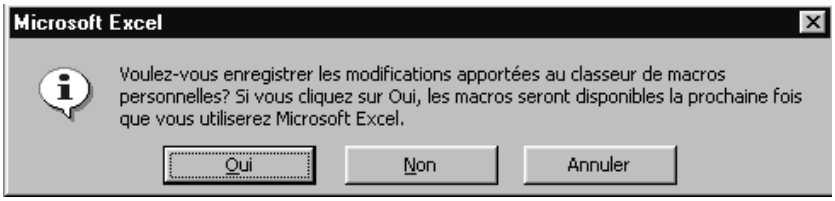
Le Classeur de **macros personnelles** est un classeur masqué qui n'existe que si on y stoque au minimum une macro. Il se nomme **PERSO.XLS** et se trouve rangé dans le répertoire de démarrage ...**EXCEL\XLOUVRIR**



On peut le visualiser par le menu **Fenêtre/Afficher.** Dans ce cas il est conseillé ensuite ultérieurement de le re-masquer par le menu **Fenêtre/Masquer.**



Il est à noter que toute modification du classeur **PERSO.XLS** se traduira à la sortie de Excel par un message du type



N.B : Si on ne stocke pas la macro au bon endroit lors de sa création, on peut ensuite la transférer d'un classeur à un autre via l'éditeur visual basic

---

Portée d'une macro:

Le classeur la contenant n'est pas ouvert		macro non disponible
Le classeur la contenant est ouvert (que ce soit de façon automatique ou manuelle)	si associée à un bouton	Disponible dans la feuille contenant ce bouton
	si associée à un outils	Disponible si outils affiché
	si incorporée dans un menu personnel	Toujours Disponible
	si associée à une combinaison de Touches	Toujours Disponible

Voir exercice "Macro enreg formatage" dans le support TP

Voir exercice "Macro enreg Datej" dans le support TP

L'ENVIRONNEMENT DE VBA

L'éditeur de visual basic :

A la différence de toutes les versions précédentes, la feuille contenant le code des macros fait toujours partie du classeur, mais n'est plus affichée en clair dans le classeur.

On peut visualiser ces feuilles en demandant le menu

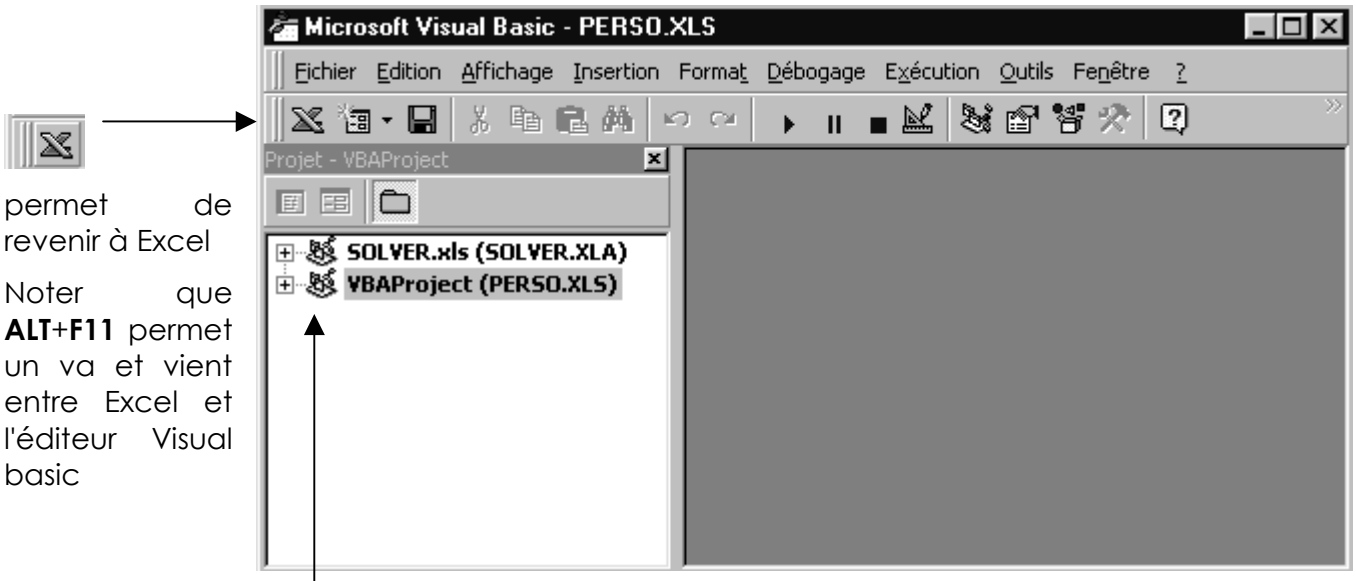
Outils / macro / macros

puis après avoir sélectionnée la macro voulue on demande **Modifier**.

On peut également aller dans l'éditeur visual basic en demandant le menu

Outils / macro / visual basic editor voire la combinaison **ALT** + **F11**

On se retrouve dans l'environnement de l'éditeur visual basic :



permet de revenir à Excel

Noter que **ALT+F11** permet un va et vient entre Excel et l'éditeur Visual basic

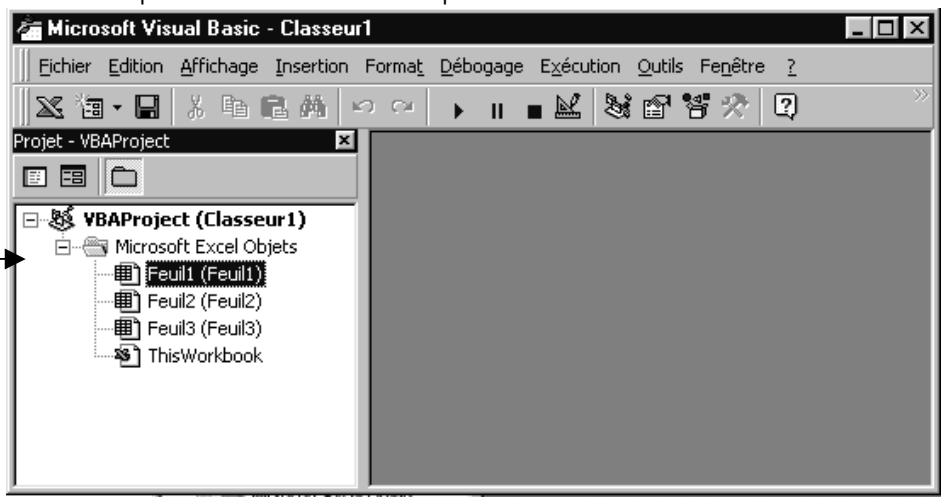
Fenêtre explorateur de Projet :

Le terme de projet est plus large que celui de macro car un projet représente non seulement des macros, mais aussi des feuilles excel classiques dans lesquelles on pourrait avoir à saisir de l'information, par exemple, et , de manière générale, tout ce qui peut constituer un ensemble fonctionnel pour une application donnée

Pour qu'un projet apparaisse il faut que le document auquel il est attaché soit ouvert dans l'application Excel

Ici deux projets apparaissent : la macro complémentaire **Solveur** et le classeur de **macro personnelles**

Voici l'éditeur avec uniquement un classeur par défaut de 3 feuilles ouvert dans EXCEL



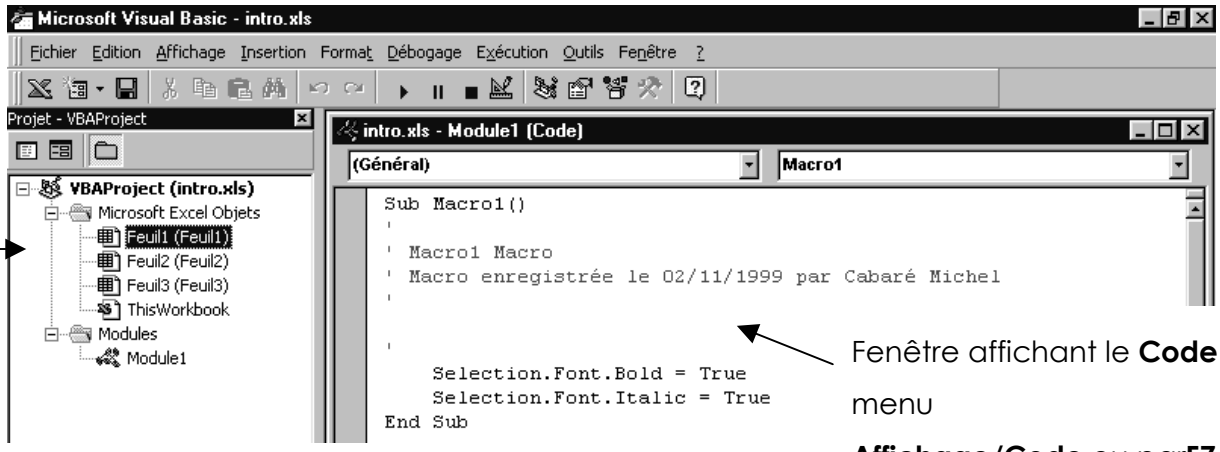
Fenêtre Explorateur de Projet **VBAProject**

menu

Affichage/Explorateur projet ou touches **CTRL + R**

Fenêtre Code :

permet d'éditer le code Vba proprement dit, à l'aide d'outils particuliers, comme une coloration syntaxique, un déboguer et des aides à l'écriture



Projet **intro.xls**

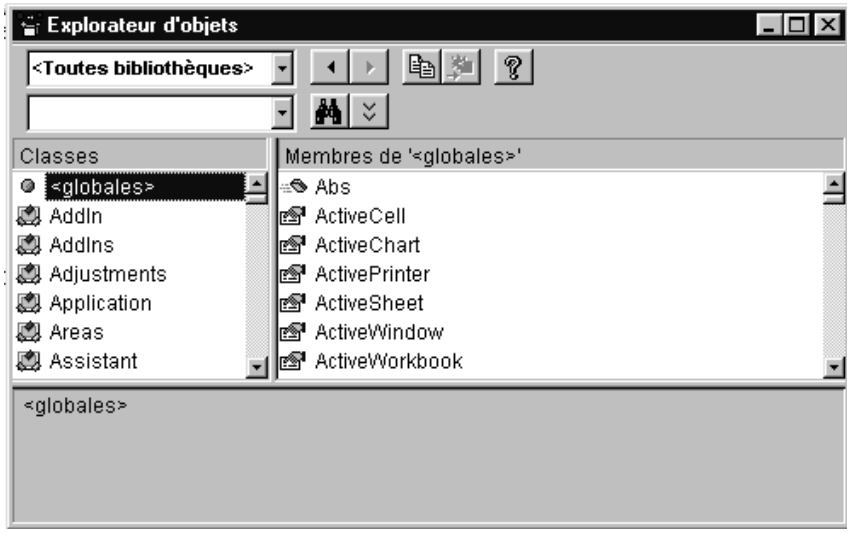
Fenêtre affichant le **Code**

menu

Affichage/Code ou par**F7**

Fenêtre Explorateur d'objet :

permet de référencer de manière automatique les propriétés, méthodes ou évènements s'appliquant aux objets utilisables



Fenêtre de **Explorateur d'objets**

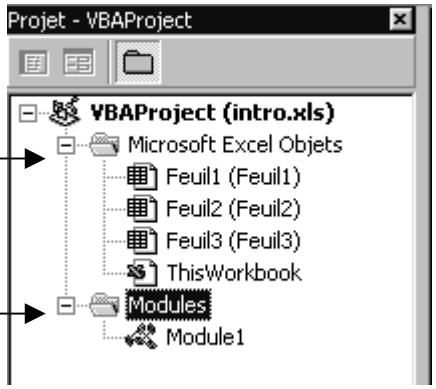
menu

Affichage/Explorateur objet ou touches **F2**

FENETRE EXPLORATEUR DE PROJET

Présentation :

La feuille contenant le code des macros fait toujours partie du classeur, mais n'est plus affichée en clair dans le classeur.



Dossier **Microsoft Excel Objets**

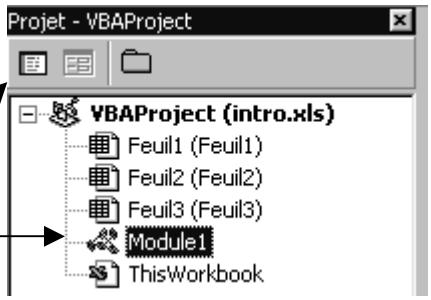
contient les éléments du classeur dans lequel le projet est stocké

Dossier **Modules**

contient les modules existants dans ce classeur, ici **Module1** (chaque enregistrement de macro crée un module appelé **Module n°x**)

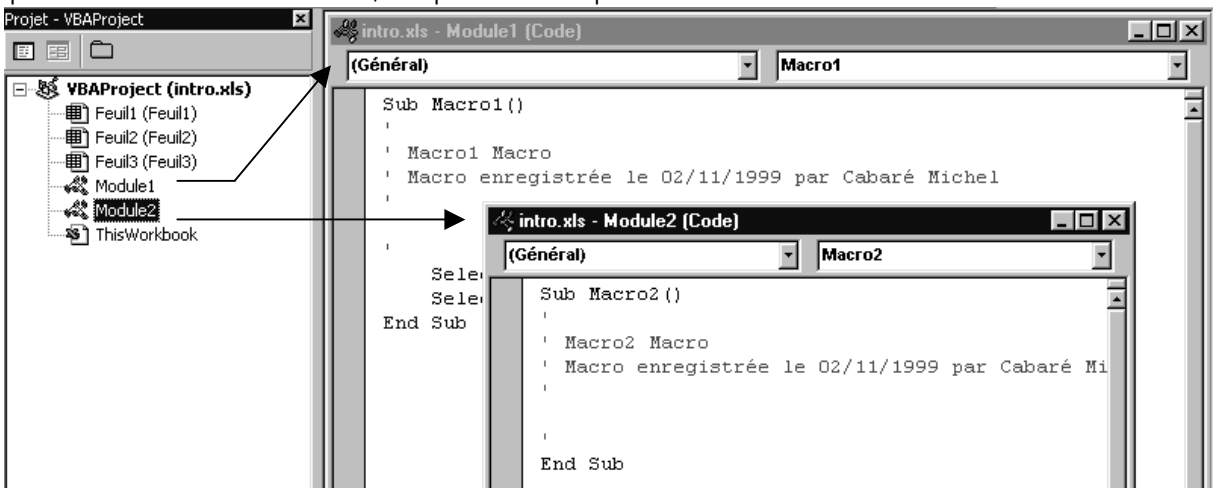
Lorsque l'on est positionné sur un module, un clic sur l'outils

ou un double clic sur le nom du module



Ouvre une fenêtre affichant le code du module

Si plusieurs modules existent, on peut ouvrir plusieurs fenêtres



Manipulation de modules :

L'objectif est de pouvoir supprimer, déplacer, dupliquer les modules dans lesquels on à une macro qui nous intéresse, comme par exemple pour mettre dans le classeur de macro personnelles une macro que l'on aurait fini de tester, ou effectuer un duplicata d'une macro pour la modifier…

Créer un module

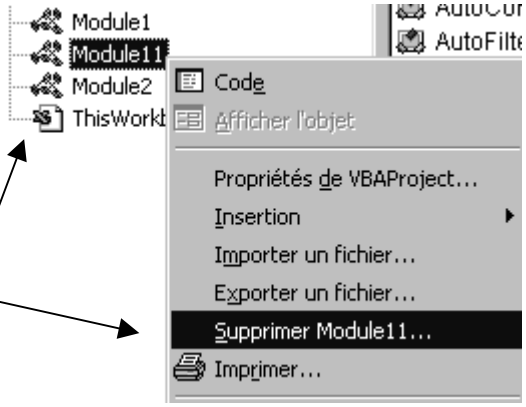
Il suffit simplement de demander dans l'éditeur visual basic le menu

Insertion / Module

Supprimer un module

Il suffit simplement de demander le menu contextuel en cliquant sur le module à supprimer, et demander le menu **Supprimer …**

ici avec Module11

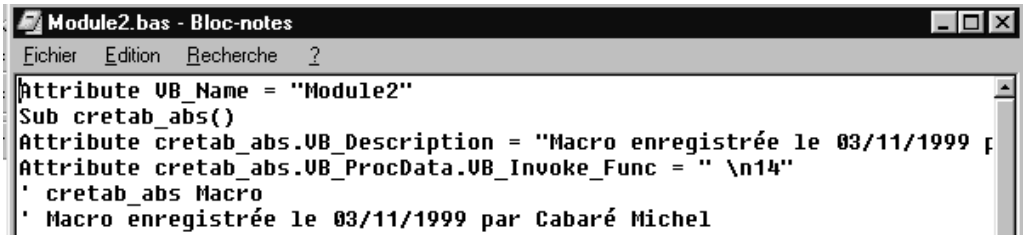


Exporter / Importer un module

Un module peut être stocké de manière autonome dans un fichier **xxx.bas** de type texte, grâce au menu

Fichier / Exporter un fichier et respectivement **/Importer un fichier**

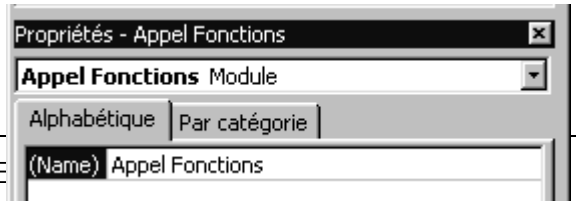
ce qui, visualisé dans un éditeur donnerais par exemple



Changer le nom d'un module

Les nom de module sont donnés de manière automatique, et on peut facilement les renommer en demandant, une fois que l'on est positionné sur le module à renommer, l'outils propriété

qui amène la fenêtre propriété dans laquelle on modifie le nom…



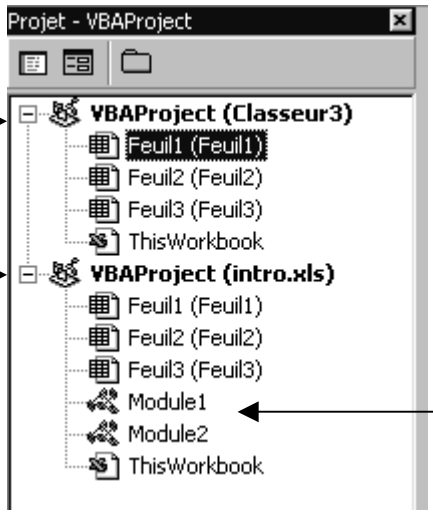
Copier, déplacer un module

Il faut tout d'abords dans EXCEL ouvrir les deux classeurs entre lesquels je souhaite effectuer mes transferts de modules

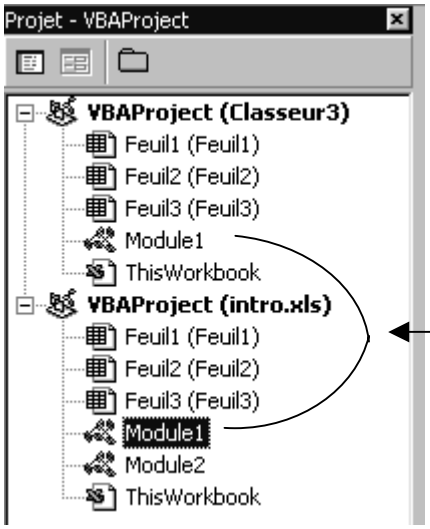
Cette opération effectuée, on se retrouve dans l'Editeur Visual Basic avec un écran du type suivant :

Fenêtre Explorateur de Projet **VBAProject**

affichant autant de projets que de classeurs ouverts dans Excel **:** ici **2**



Les modules apparaissent clairement, et on va les manipuler comme les fichiers dans l'explorateurs windows



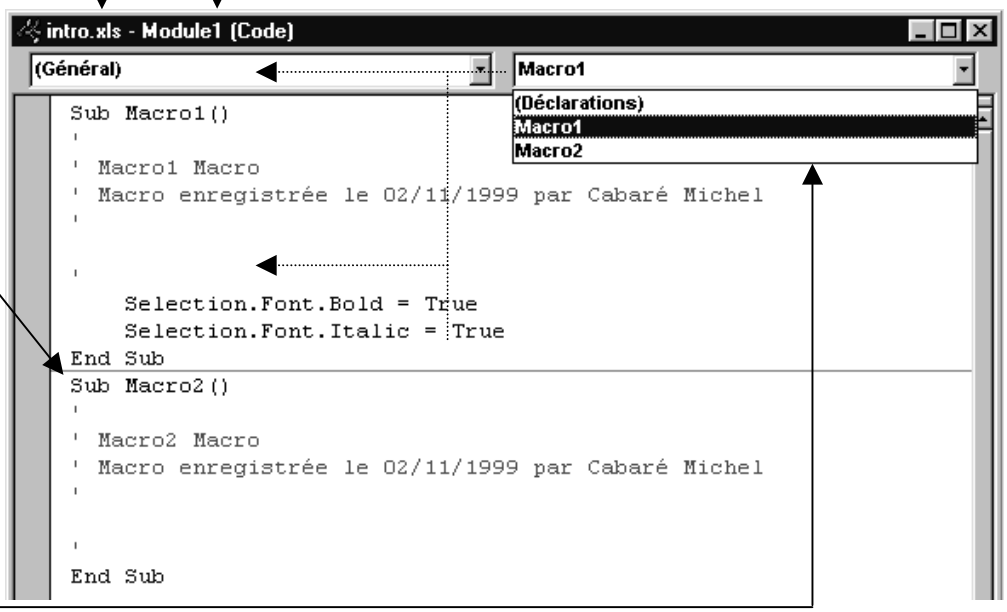
Faire glisser un module le **duplique**

FENETRE CODE

Présentation :

Une fenêtre code est toujours attachée à l'un des modules standard vba apparaissant dans l'explorateur de projet, ou plus généralement à l'un quelconque des modules (feuille…)

La fenêtre **Code** affiche l'ensemble des procédures constituant le module auquel elle est attachée. le **nom du module** paraît dans la barre de titre après le **nom de projet**



toutes les procédures sont séparées par un trait gris, et on peut y accéder directement en haut à droite

Ainsi qu'à la zone de déclaration générale du module…

Manipulation de code :

tous les principes standard du traitement de texte sont valables, y compris les copier, couper et coller

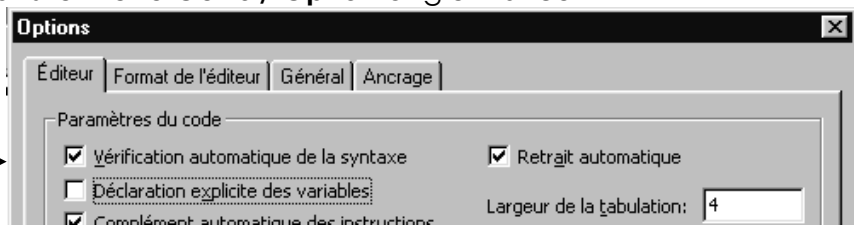
On peut sélectionner des lignes complète en amenant la pointe de la souris sur la gauche de manière à ce qu'il prenne la forme d'un flèche orientée vers le droite. A ce moment là :

- un clic sélectionne toute la ligne
- double clic sélectionne toute la procédure

Vérificateur, analyseur de syntaxe :

Une vérification automatique de la syntaxe est installée par défaut, et on peut l'enlever dans le menu **Outils / Option** onglet **Editeur**

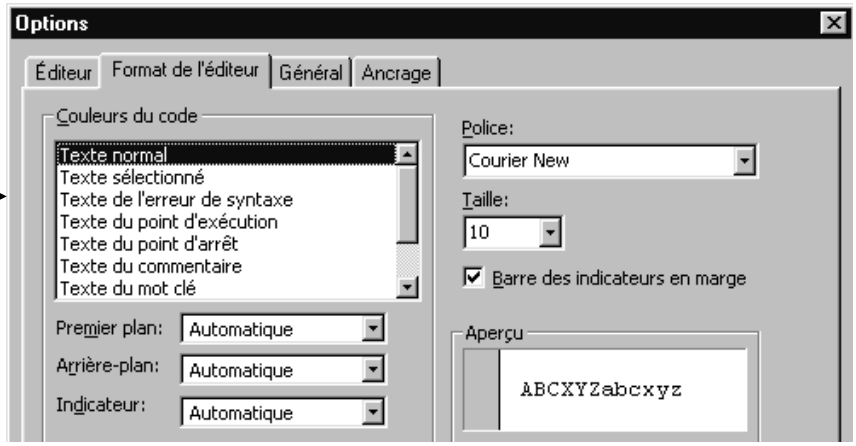
Il vaut mieux la garder →



Une coloration syntaxique est présente et permet à un utilisateur de mieux se repérer.

dans le menu **Outils / Option** onglet **Format de l'éditeur**

Un certain nombre de coloration sont déjà en vigueur →



Par défaut on à :

erreur de syntaxe	premier plan: rouge
commentaire	premier plan: vert
mot clé	premier plan: bleu

On pourrait aussi demander pour améliorer l'affichage :

mot clé	premier plan: blanc
	arrière plan: bleu
identificateur	premier plan: bleu
	arrière plan: bleu ciel

1° macro en VBA :

écrire de toute pièce la macro suivante (qui ne fait rien mais qui existe !)

```
Sub ouf()

End Sub
```

RELATIF OU ABSOLU ?

Pendant l'enregistrement :

Lorsque l'on enregistre une macro, on peut soit l'enregistrer avec des références relatives, soit l'enregistrer avec des références absolues.

Il suffit après avoir démarré l'enregistrement, de cliquer sur l'outils:

références relatives qui devient clair lorsqu'il est activé

Indépendamment de comprendre complètement l'écriture générée, on se rends compte que l'enregistrement en relatif part à partir de la cellule ou l'on est au moment où on demande l'exécution de la macro, alors qu'un enregistrement en absolu part toujours de la cellule où l'on était lors de l'enregistrement de la macro !

Ainsi si l'on fait une macro enregistrée ayant pour but de supprimer la colonne ou l'on se trouve, les résultat vont varier selon :

- Le mode d'enregistrement de la macro : **Relatif / Absolu**
- Si en cours d'enregistrement on a effectué une sélection de cellule ou non (déplacement)

Voir exercice "Relatif ou Absolu ?" dans le support TP

COMPLETER UNE MACRO

A partir d'une macro existante :

par exemple une macro appelé **creation** construisant le tableau suivant:

	A	B	C	D
1		1999	2000	Total
2	1° trim			0
3	2° trim			0
4	3° trim			0
5	4° trim			0

Compléter cette macro par un enregistrement :

Pour compléter avec une macro enregistrée nommée **suite** il faut :

1. se placer dans le contexte exact dans lequel la macro que l'on souhaite compléter s'arrête
2. créer de toute pièce une nouvelle macro avec la "suite" des opérations, par l'enregistreur de macro
3. Aller dans l'éditeur visual basic et afficher les modules des deux macros

N.B: attention, les deux macros ne seront pas forcément dans le même module, et il sera donc dans ce cas nécessaire d'ouvrir une fenêtre pour chaque module !

4. Copier tout le code à l'exception des mots-clé **Sub** et **EndSub**
5. le coller dans la macro à compléter juste avant le mot-clé **EndSub**
6. enregistrer

Voir exercice "Compléter un Enregistrement" dans le support TP

NOTIONS DE P.O.O.(VOCABULAIRE)

Objet - Propriété - Méthode - Evènement...

Objet:

Pour illustrer le concept d'objet, avec ses propriétés et ses méthodes, prenons un exemple de la vie courante . Un objet est une chose, au sens courant du terme, par exemple une voiture. Et un objet peut contenir d'autres objets, par exemple un moteur, une portière...

Collection:

Si les objets contenus à l'intérieur d'un objet sont de même type, on parlera alors de collection pour l'ensemble des objets de ce type.

Les éléments d'une collection sont identifiés par numéro ou par nom

Méthode :

On peut définir les actions possibles avec cet objet, grâce cette fois-ci, non pas à des adjectifs, mais à des verbes. C'est ce que l'on appelle des méthodes. Ainsi avec une voiture, on peut démarrer, accélérer, freiner, tourner... L'ensemble des méthodes d'un objet représente toutes les actions que l'on peut effectuer avec cet objet.

On ne peut faire qu'une seule chose avec une méthode, c'est l'appliquer sur l'objet souhaité. Cependant les méthodes possèdent souvent des paramètres (arguments) qui précisent la façon dont on veut que la méthode s'applique. Ainsi la méthode Freiner applicable à l'objet voiture pourrait se paramétrer par doucement, sec, pile...

Propriété :

Pour décrire un objet on utilise des adjectifs qui le caractérise, ce sont les propriétés de cet objet : couleur, année, marque, nombre de siège... L'ensemble des propriétés d'un objet représentent tous les aspects de cet objet.

On peut s'en servir soit pour avoir une information sur l'objet (lecture) soit pour modifier l'objet (écriture). Cependant certaines propriétés sont en lecture seule, c'est à dire non modifiables, par exemple l'année de construction de notre voiture.

Evènement :

Un événement est une action qui peut être détectée sur un objet, comme un clic souris, ou la frappe d'une touche. l'idée en général consiste à vouloir écrire un code spécifique vba en réponse à cet événement.

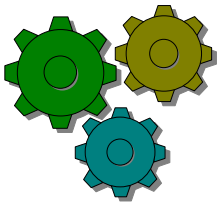
Un événement peut être déclenché par une action de l'utilisateur mais aussi par le système lui même

Exemple de modèle objet :

Objet VOITURE

(conteneur de moteur, siège…)

Propriétés de l'objet VOITURE :	**nom**	**type**
	Couleur	Lect/écrit
	Année	Lecture seule
	Marque	Lecture seule
	Nb sièges	Lect/écrit
	Vitesse	Lect/écrit
Méthodes de l'objet VOITURE :	**nom**	**paramètres**
	Démarrer	Vite, lent,norm
	Accélérer	Vite, maxi
	Freiner	Sec, Doux,pile
	Tourner	gauche,Droite
Méthodes de l'objet VOITURE :	**nom**	**renvoit type**
(Particulières)	Sièges	Sièges (coll)
		Siège (obj)
	Moteur	Moteur



Objet MOTEUR

(conteneur de …)

Propriétés de l'objet MOTEUR :	**nom**	**type**
	Puissance	Lecture seule
	Nb cylind	Lecture seule
	Vidange	Lect/écrit

Pour récupérer la date de vidange, il faudrait écrire :

Date = Voiture.Moteur.Vidange

Pour modifier la date de vidange , il faudrait écrire :

Voiture.Moteur.Vidange = 01/04/96

Pour représenter l'ensemble des sièges de la voiture on écrirait :

Voiture.Sièges (c'est à dire une collection)

alors que pour représenter un objet de type siège (le 2°)on écrirait :

Voiture.Siège(2)

ET EXCEL DANS TOUT ÇA ?

Les Objets Excel

Un objet est un élément constitutif d'une Application et que l'on peut contrôler par un langage de programmation.

Les objets d'excel vont permettre de manipuler des "grandeurs réelles " telles qu'elles existent dans Excel, par exemple une FEUILLE est un objet Excel, mais aussi un CLASSEUR, ou une LIGNE, ou un GRAPHIQUE...ou une simple CELLULE.

Ainsi, un Objet Excel est particulier et inutilisable ailleurs: il est vraiment impensable d'utiliser la notion de CELLULE à l'intérieur de Word ou de Access n'est-ce-pas?

C'est pourquoi on parle de VBA pour Excel, de VBA pour ACCESS par opposition à VB lorsque il s'agit du langage de programmation non dédié à un logiciel spécifique.

Dans un code VBA, il faut identifier l'**objet** avant de pouvoir lui appliquer une **méthode** ou pouvoir lire/modifier une de ses **propriétés**

Il y à beaucoup d'objets différents dans EXCEL allant du simple rectangle au graphique ou au tableau croisé dynamique ! Certains objets peuvent contenir d'autres objets , ainsi l'objet **Workbook** (classeur) peut contenir plusieurs objets de type **Worksheet** (feuille), qui contiennent eux-mêmes d'autres objets de type **Range** (cellule).

Les Collections Excel

Une collection est un objet contenant plusieurs autres objets, (généralement de même type). On repère une collection généralement au "**S**" à la fin du nom de l'objet composant la collection

Dans Excel, par exemple l'objet **Workbooks** contient tous les objets **Workbook**

Les éléments d'une collection peuvent être identifiés par numéro ou par nom. Ainsi

Workbooks(1) repère le premier classeur ouvert dans Excel

alors que

Workbooks("toto.xls") repère le classeur ouvert nommé "toto.xls"

et l'écriture globale

Workbooks repère tous les classeurs ouvert dans Excel

Les Méthodes dans Excel

Chaque objet d'EXCEL possède un ensemble de méthodes qui représentent les actions réalisables avec cet objet.

Prenons par exemple l'objet **Workbook** (Classeur) , ses caractéristiques sont donc celles d'un fichier EXCEL , et voici une liste partielle de ses méthodes : **Activate** (Activer), **Close** (Fermer)...

Les méthodes dépendent de l'objet sur lequel elles s'appliquent, et peuvent recevoir des arguments précisant la façon dont on veut qu'elles s'appliquent. Par défaut certaines valeurs sont données aux arguments.

<u>Syntaxe</u> : On sépare l'objet et sa méthode par un "." (point)

<u>Syntaxe</u> : On sépare plusieurs arguments par une "," (virgule)

méthode sans paramètres :

Ici on ferme tous les classeurs d'Excel

```
Workbooks.Close
```

méthode Close s'appliquant à un objet de type Workbook

(Si l'un des classeurs ouverts a été modifié, Excel affiche l'invite et la boîte de dialogue permettant à l'utilisateur d'enregistrer ces modifications.

méthode avec paramètres :

Ici on ferme le 1° classeur ouvert dans Excel et on ignore toutes les modifications qui lui ont été éventuellement apportées

```
Workbooks("1").Close SaveChanges:=False
```

méthode Close idem ci-dessus

paramètre donné

Les Propriétés dans Excel

Chaque Objet d'EXCEL possède un ensemble de propriétés qui le décrit. Prenons par exemple l'objet **Workbook** (Classeur) , ses caractéristiques sont donc celles d'un fichier EXCEL , et voici une liste partielle de ses propriétés : **Name** (Nom), **Path** (CheminAccès) Etc ...

On peut faire deux choses avec les propriétés, les "lire", c'est à dire connaître leur valeur ou les "définir", c'est à dire définir leur valeur (si la syntaxe le permet)

Les propriétés donc dépendent de l'objet qu'elles définissent, elles sont de natures variables, Chaîne de caractère (nom de fichier) valeurs numériques (largeur de cellule)

<u>Syntaxe</u> : On sépare l'objet de sa propriété par un point "."

lecture de valeurs de propriété :

Ici on lit le nom du 1° classeur ouvert dans Excel

```
val= Worksbook(1).name
```

propriété name s'appliquant à un objet de type Workbook en lecture seule

(Avec val variable de type chaîne de caractère)

Ici on lit le nom de la première feuille du classeur actif

```
val= Worksheet(1).name
```

propriété name s'appliquant à un objet de type Worksheet en lecture seule

(Avec val variable de type chaîne de caractère)

écriture d'une valeurs dans une propriété :

Ici on masque la feuille de calcul 1 dans le classeur actif.

```
Worksheets(1).Visible= False
```

propriété s'appliquant à un objet de type Worksheet en lecture / écriture

(Avec False constante)

Propriétés particulières

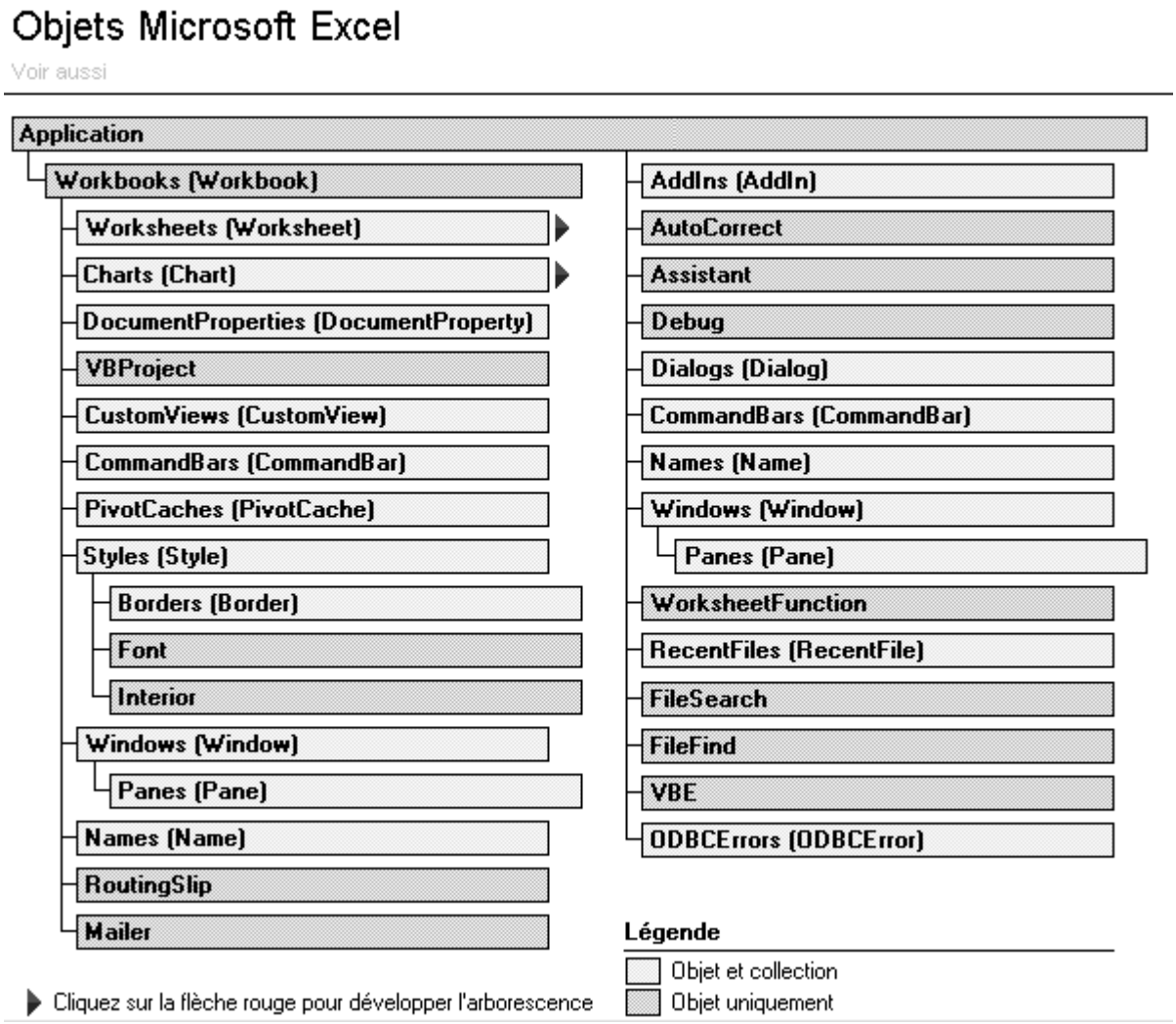
Il existe des propriétés qui renvoient des objets au lieu des caractéristiques de l'objets sur lequel elles s'appliquent

Dans ce cas on peut traiter la propriété comme un objet, et lui appliquer notamment une propriété ou une méthode à son tour (exactement les mêmes que celles que l'on peut appliquer sur le type d'objet renvoyé)

HIÉRARCHIE DES OBJETS

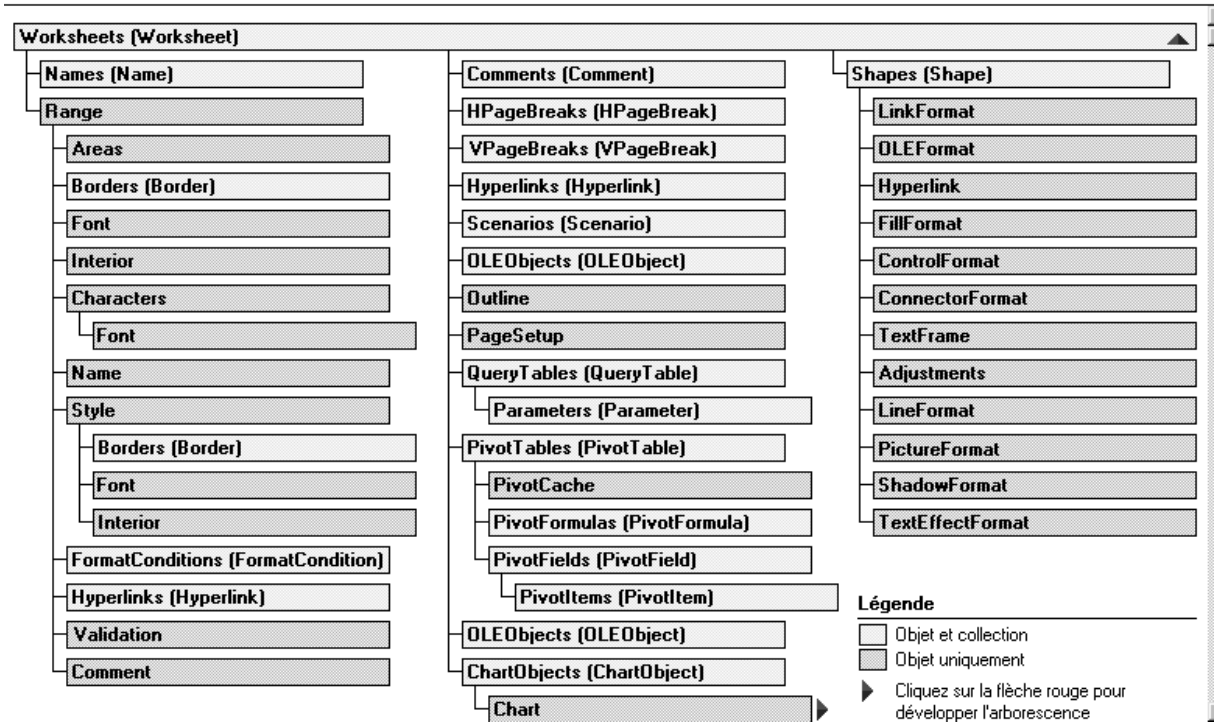
Vue d'ensemble :

Comment connaître les noms des différents objets dans excel ?, savoir qui englobe qui, et quelles sont ses méthodes et propriétés applicables ?



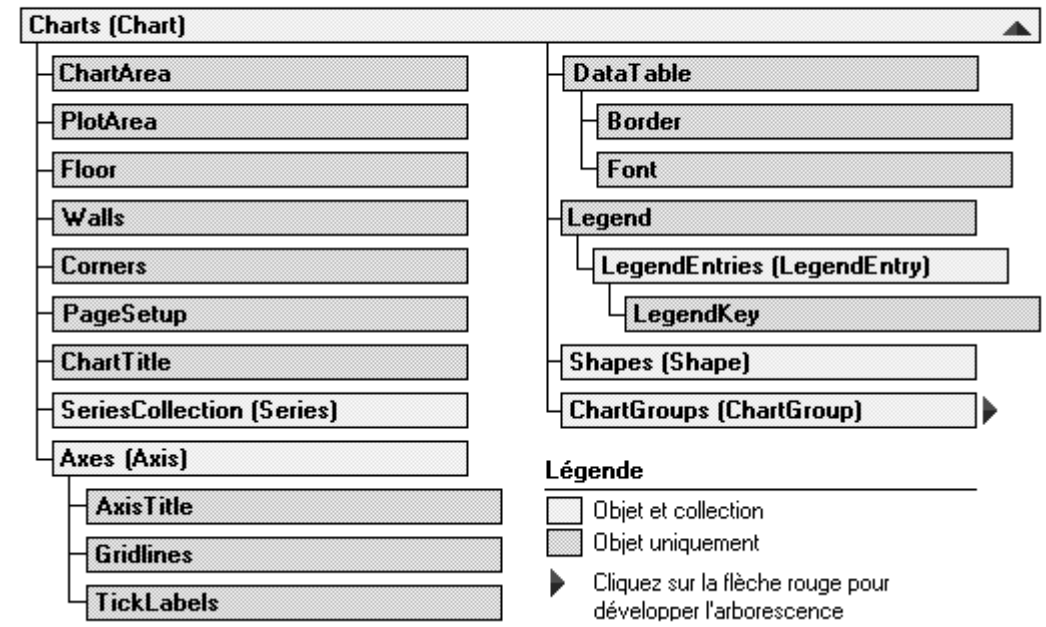
Objets Microsoft Excel (Feuille de calcul)

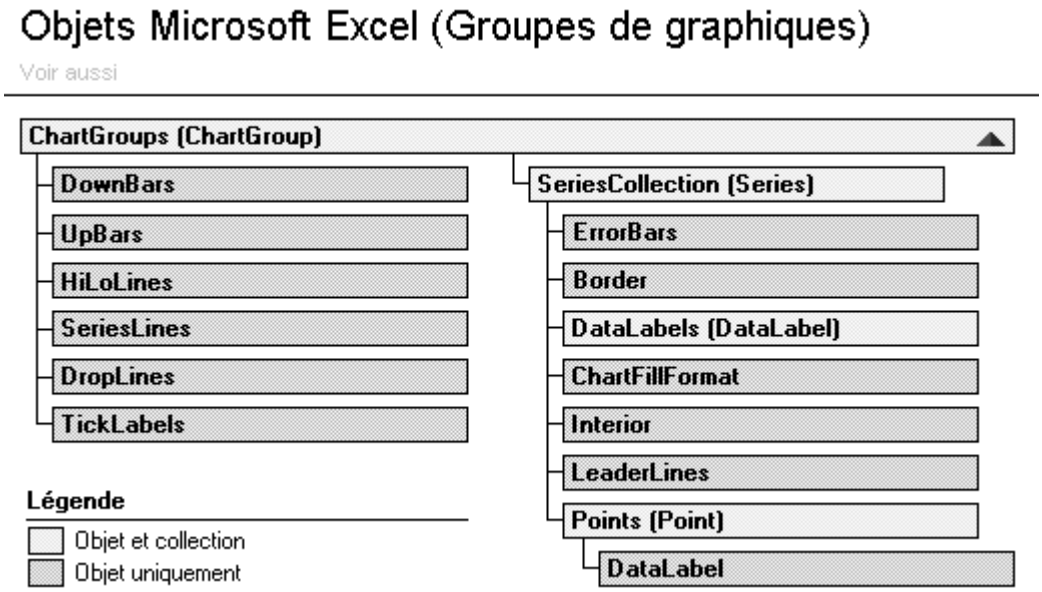
Voir aussi



Objets Microsoft Excel (Graphiques)

Voir aussi





Reste le problème de l'écriture dans la macro

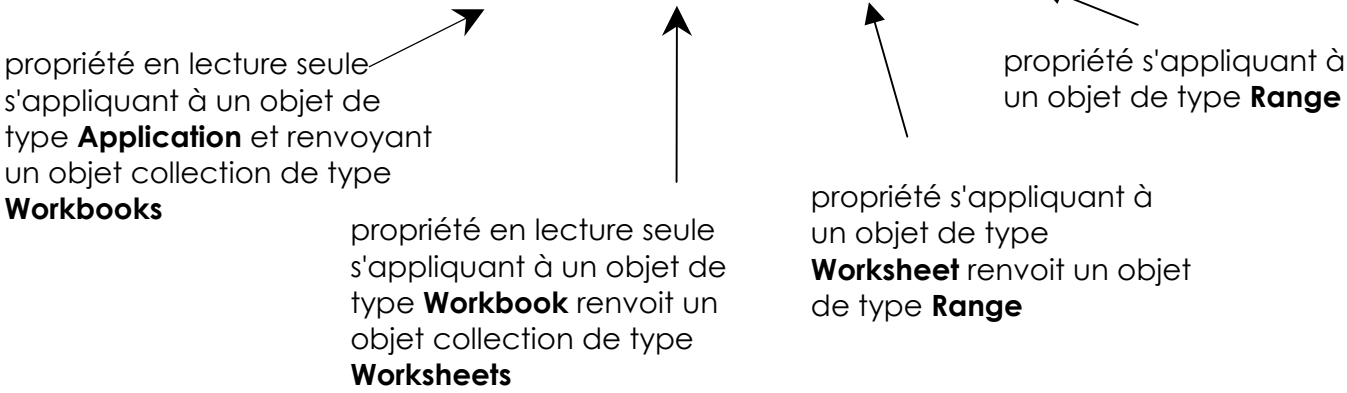
RÊGLES D'ÉCRITURE

Ecriture complète :

On peut bien sûr utiliser les organigrammes précédents pour connaître la structure d'un objet. Pour manipuler les propriétés et les méthodes d'un objet on peut faire référence à tous les objets qui font partie du chemin hiérarchique.

La voie hiérarchique entre un objet conteneur et un objet contenu (ou une méthode renvoyant un objet contenu) s'obtient par le "." (point), on pourrait écrire pour atteindre la valeur de la plage A1 de la première feuilles du premier classeur ouvert dans Excel

```
Application.Workbooks(1).Worksheets(1).Range("A1").Value = "ouf"
```



Heureusement un certains nombre de références peuvent être faites par défaut.

Référence implicite à Application :

à partir du moment où l'on sait que le code ici est exécuté toujours depuis EXCEL, (comme c'est généralement le cas) il n'est pas nécessaire de faire explicitement référence à l'application EXCEL. On peut donc écrire (si la syntaxe le permet) :

```
Workbooks(1).Worksheets(1).Range("A1").Value = "ouf"
```

Référence implicite au Classeur :

Dans EXCEL on peut avoir plusieurs classeurs ouvert à un moment donné, cependant un seul est actif à un instant précis. Ainsi au lieu de référencer un index dans la collection Classeurs, on peut prendre le classeur Actif.

On pourrait donc écrire ce qui reviendrait au même si le classeur voulu est actif :

```
ActiveWorkbook.Worksheets(1).Range("A1").Value = "ouf"
```

On pourrait aussi écrire ce qui reviendrait au même s'il y a un seul classeur ouvert (et donc actif):

```
Worksheets(1).Range("A1").Value = "ouf"
```

Référence implicite à la feuille active :

De façon similaire on pourrait écrire ce qui reviendrait au même si c'est la feuille 1 qui est active :

```
Activesheet.Range("A1").Value = "ouf"
```

Voire s'il n'y a qu'une seule feuille ou qu'il n'y ait pas d'ambiguïté :

```
Range("A1").Value = "ouf"
```

A quel niveau écrire :

Cela dépend donc notamment :

- **de la sécurité que l'on exige.**
- **de la syntaxe requise par les objets que l'on emplois (objets requis ou non)**

ATTENTION: le code se trouve dans le classeur **macro vba us v8 cours .xls** dans le projet **vbaxlsv8cours**, dans un module nommé **Regle_Ecriture**

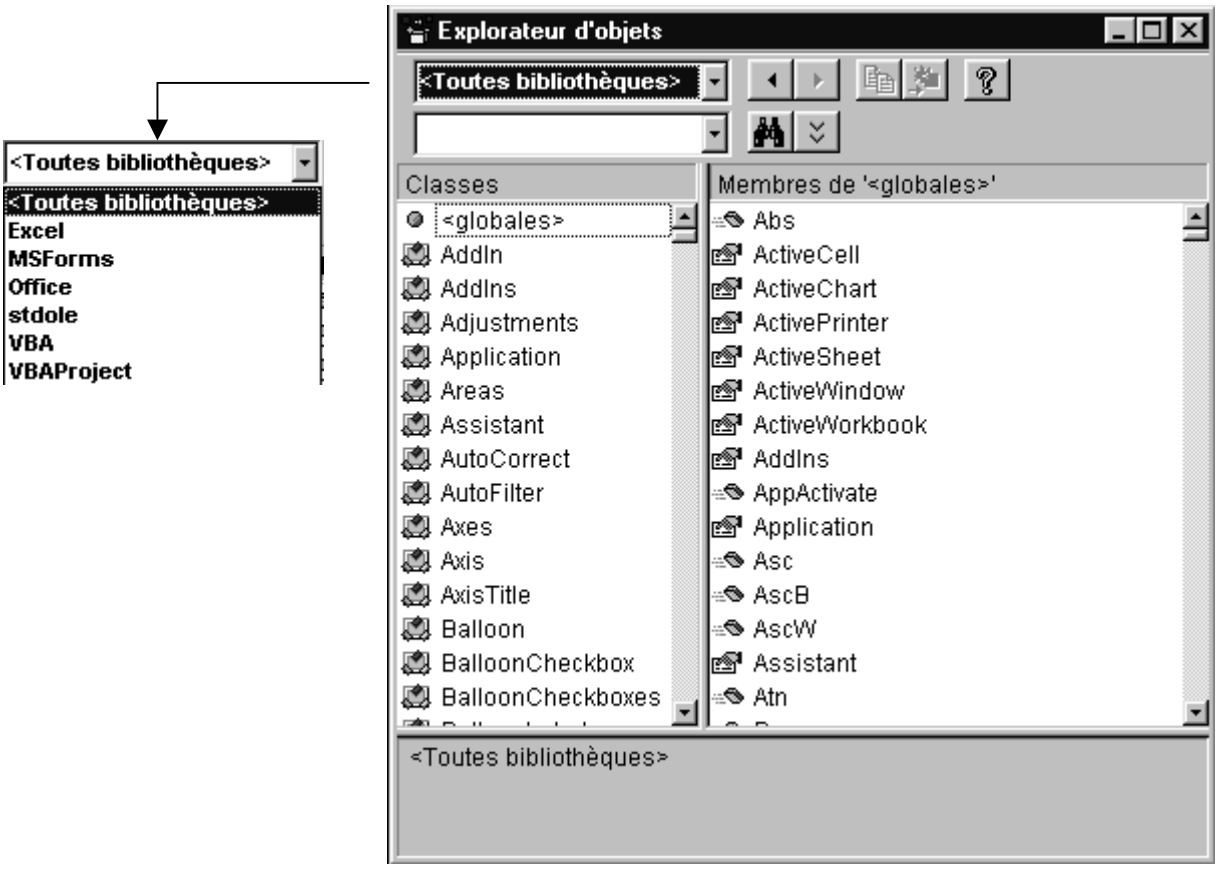
FENETRE EXPLORATEUR D'OBJET

Présentation :

L'explorateur d'objet est indispensable lorsque l'on veut trouver pour un objet donné la liste de tous les membres associés à cet objet, c'est à dire les méthodes, les propriétés, les événements ou les constantes qui y sont associées.

Le fichier contenant ces associations se nomme **excel8.olb**

la fenêtre de l'explorateur d'objet se présente ainsi



QUITTER EXCEL

Objectif & analyse

On veut quitter excel en cliquant sur un bouton, après confirmation de la part de l'utilisateur. Le déroulement donc devrait être le suivant :

Demander si on veut réellement quitter
Si Reponse=Oui **Alors**
    Quitter
Sinon
    Rien
Fin si

Eléments présentés :

- Déclaration de variable.
- Utilisation de constantes
- Instruction **If**...**then**...**else**...**endif**
- Fonction **Msgbox**
- Méthode **Quit** sur Objet **Application**

Code :

```
Option Explicit
Sub quitter_excel()          'Cette procédure demande confirmation à
                             'travers une boite de dialogue avant de
                             'quitter excel. Seuls OUI ou NON sont
                             'offertcomme réponse à l'utilisateur

  Dim Reponse As Integer


  Reponse = MsgBox("Voulez vous réellement quitter excel ?", _
                   vbYesNo, " Confirmez svp!")

  If Reponse = vbYes Then     'Si utilisateur confirme par Oui ...
    Application.Quit
  End If


End Sub
```

On peut noter que MsgBox accepte une syntaxe plus simple à partir du moment ou l'on ne s'intéresse pas au résultat de la réponse donnée par l'utilisateur après l'affichage

Supposons un excès de politesse dans la macro permettant de quitter excel, il faut saluer l'utilisateur par le code

```
MsgBox "Au Revoir "
```

L'utilisateur ne peut que valider pour indiquer la prise de connaissance du message, il n'y a pas d'autres choix

ATTENTION: le code se trouve dans le classeur **macro vba us v8 cours .xls** dans le projet **vbaxlsv8cours**, dans un module nommé **Quitter_Excel**

SE PLACER EN ABSOLU

Objectif & analyse

Pouvoir se déplacer par rapport à la feuille dans une cellule précise, (et non plus par rapport au point de départ de la macro).

Eléments présentés :

- propriété **Cells**(L,C)
- propriété **Range**("réf")
- méthode **Activate** sur objet **Range**

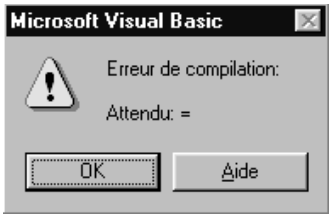
Propriété Cells :

A noter que les repères sont plus facilement utilisables lorsque on est habitué aux coordonnées de type LxCy…(au lieu de A2…)

```
Sub Placer_abs()
   Cells(1, 2)
End Sub
```

avec la cellule…

cela ne suffit pas et Vb génère un message d'erreur →



on doit dire quoi faire avec l'objet Range que la propriété Cells renvoie

```
Sub Placer_abs()
   Cells(1, 2).Activate
End Sub
```

Active ligne 1 colonne 2 (soit B1)

Noter que l'écriture

```
Application.Cells(1, 2).Activate
```

serait identique (mais l'omission de l'objet Application est autorisée…)

et que les écritures suivantes

```
Application.ActiveSheet.Cells(1, 2).Activate
```

ou

```
ActiveSheet.Cells(1, 2).Activate
```

sont valables, elles s'appuient sur le fait que Cells peut s'appliquer sur un objet de type **WorkSheet**…

L'utilisation de variables est tout à fait possible

```
Sub placer_abs_2()
Dim lig, col As Integer

   lig = 10
   col = 5
   Cells(lig, col).Activate
End Sub
```

Propriété Range :

L'objectif n'est pas différent de celui atteint par la propriété Cells, mais la notation diffère sensiblement, Il doit s'agir d'une référence en style A1. Cet argument peut inclure l'opérateur de la plage (deux points), l'opérateur d'intersection (espace) ou l'opérateur d'union (virgule). Il peut aussi inclure des signes dollars mais ils sont ignorés. Vous pouvez utiliser un nom défini en local dans n'importe quel endroit de la plage

on doit dire quoi faire avec l'objet Range que la propriété Range renvoie

```
Sub Placer_abs()
   Range("B4").Activate
End Sub
```

Active B4 (soit ligne 4 colonne 2)

Noter que l'écriture

```
Range("toto").Activate
```

marcherait si une cellule nommée "toto" existe dans la feuille

Noter que l'écriture

```
Application.Range("B4").Activate
```

serait identique (mais l'omission de l'objet Application est autorisée…)

et que les écritures suivantes

```
Application.ActiveSheet.Range("B4").Activate
```

ou

```
ActiveSheet.Range("B4").Activate
```

sont valables, elles s'appuient sur le fait que range peut s'appliquer sur un objet de type **WorkSheet**…

ATTENTION: certains exemple de ces deux chapitres se trouvent dans le classeur **macro vba us v8 cours .xls** dans le projet **vbaxlsv8cours**, dans un module nommé **Déplacement**

SE DÉPLACER EN RELATIF

Objectif & analyse

Pouvoir se déplacer par rapport à la cellule dans laquelle on se trouve au moment où la macro s'exécute, (et non plus par rapport à des coordonnées de cellule).

Eléments présentés :

- propriété **ActiveCell**
- propriété **Offset**(x,y) sur objet **Range**
- méthode **Activate** sur objet **Range**

Propriété ActiveCell :

Cette propriété renvoie un objet Range qui représente la cellule active de la fenêtre active

Toutes ces écritures sont équivalentes

```
ActiveCell
Application.ActiveCell
ActiveWindow.ActiveCell
Application.ActiveWindow.ActiveCell
```

Propriété Offset(x,y) :

On peut Utiliser la propriété **Offset** pour déplacer la cellule active. La procédure suivante montre comment insérer du texte dans la cellule active de la plage sélectionnée puis déplacer la cellule active 1 d'une cellule vers la droite sans modifier la sélection

A noter que le programme suivant le marcherait pas, pourquoi ?

```
Sub deplacer_rel()
   ActiveCell.Offset(1, 0)
End Sub
```

Que faire avec cet objet range ?…

Cela marchera mieux ainsi :

```
Sub deplacer_rel()
   ActiveCell.Offset(1, 0).Activate
End Sub
```

…l'activer par exemple !

LIRE-ECRIRE DANS UNE CELLULE

Objectif & analyse

Pouvoir se déplacer par rapport à une cellule de départ, soit pour récupérer sa valeur (de type texte ou numérique), soit pour la modifier de façon fixe (c'est à dire une valeur de type texte ou numérique connue).

Un déroulement type pourrait être le suivant :

Lire la valeur de la cellule courante et la stocker dans une variable

Se déplacer une cellule dessous (ou y aller)

Ecrire le mot« copie » dans cette cellule

Se déplacer une cellule dessous (ou y aller)

Ecrire la valeur précédemment stockée

Eléments présentés :

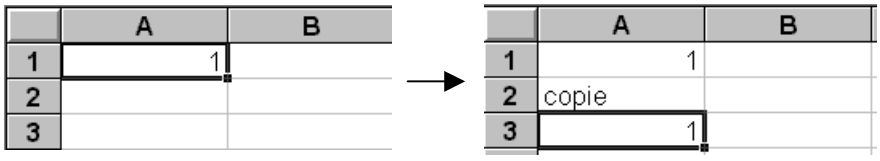
- propriété **Value** sur objet range
- propriété **Formula** et **FormulaR1C1** sur objet range

Lire écrire une valeur :

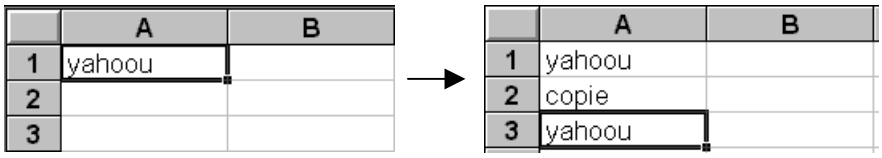
```
Option Explicit          'Oblige la déclaration de toute variable

Dim Contenu As Variant
Sub lit_ecris_valeur()
   Contenu = ActiveCell.Value
   ActiveCell.Offset(1, 0).Activate
   ActiveCell.Value = "copie"
   ActiveCell.Offset(1, 0).Activate
   ActiveCell.Value = Contenu
End Sub
```

ici on est deux cellules plus bas

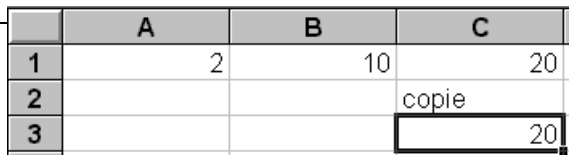
On voit que le résultat est correct pour des cellules contenant du chiffre : c

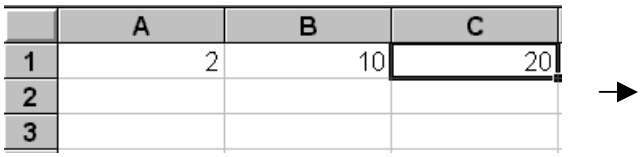


ou du texte



Mais pour une formule on ne prends que la valeur ! (ne copie pas la formule)





	A	B	C
1	2	10	20
2			
3			

On peut noter au passage que l'on pourrait obtenir la même chose mais sans déplacement de la cellule active après l'exécution de la macro

```
Sub lit_ecris_valeur_ssdp()
   Contenu = ActiveCell.Value
   ActiveCell.Offset(1, 0).Value = "copie"
   ActiveCell.Offset(2, 0).Value = Contenu
End Sub
```

ici on est dans la même cellule

Lire-écrire une formule :

Pour inscrire une formule fixe dans les cellules. Il est bon de noter que l'utilisation de Valeur est possible si la formule est "fixe"

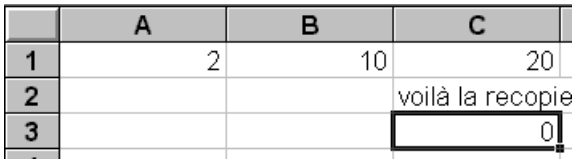
```
Proc ecris_formule_val()
    ActiveCell. Offset(1, 0).Activate
    ActiveCell.Value = "=somme(A2:A4)"
    ActiveCell.Offset(1, 0).Activate
Fin Proc
```

on écris une formule « figée » dans la cellule

Mais pour "lire" une formule dans une cellule. Il est bon de noter que l'utilisation de **Formula** ou **FormulaR1C1** est nécessaire

```
Sub lit_ecris_formule()
    Contenu = ActiveCell.FormulaR1C1
    ActiveCell.Offset(1, 0).Activate
    ActiveCell.Value = "voilà la recopie"
    ActiveCell.Offset(1, 0).Activate
    ActiveCell.FormulaR1C1 = Contenu
End Sub
```

La recopie de formule est désormais possible, et il est bon de noter que la propriété FormulaR1C1 marche aussi avec les nombre et le texte (qui peut le plus, peut le moins…)



	A	B	C
1	2	10	20
2			voilà la recopie
3			0

Différence entre **Formula** et **FormulaR1C1** : il s'agit simplement d'une différence de notation quant à l'éventuel affichage de la formule. Les deux écritures fonctionnent parfaitement, et sont indépendantes du type de coordonnées choisies par l'utilisateur dans l'affichage d'Excel.

Pour des raisons de confort, on utilisera systématiquement **FormulaR1C1**

Résumé :

accéder à une cellule précise :

Range("A2").xxx	atteint A2
Cells(1,2).xxx	atteint L1C2

accéder à une cellule relativement :

ActiveCell.xxx	cellule courante
ActiveCell.Offset(L,C).xxx	(+/-)L,(+/-)C depuis cellule courante

lire/écrire une valeur dans une cellule :

xx **= ActiveCell.Value**	**ActiveCell.Value =** xx

lire/écrire une formule dans une cellule :

xx **= ActiveCell.FormulaR1C1**	**ActiveCell.FormulaR1C1 =** "xx"

ATTENTION: le code se trouve dans le classeur **macro vba us v8 cours .xls** dans le projet **vbaxlsv8cours**, dans un module nommé **Lecture_Ecriture_Cel**

Voir exercice "Total général et Augmentation" dans le support TP

1° TANT QUE

Objectif & analyse

L'intérêt d'une macro c'est aussi de pouvoir effectuer des opérations impossibles à faire normalement dans Excel, notamment les notions de « boucles », c'est à dire de traitements dont on ne sait pas à priori combien de fois ils doivent être exécutés

Imaginons que nous souhaitions faire la somme de toutes les cellules non vides placées verticalement sous la cellule de départ de la macro, (ce qui correspondrait à la fonction somme() si on connaissait le nombre de cellule…)

Le déroulement devrait donc être le suivant :

Etant positionné sur la première cellule de la « série »
Mettre une variable à 0
Do While le cellule courante contient une valeur
    Variable = Variable + cellule courante
    Aller à la cellule suivante
Loop
Ecrire "TOTAL"
Aller 1 cellule plus bas
Ecrire la valeur de la Somme

Eléments présentés :

- Instruction **DoWhile Loop**
- Fonction **IsEmpty**
- Fonction **Not**

Boucle de parcours :

Il faut d'abords arriver à se créer le parcours de toutes les cellules, or à priori on ne sait pas combien on en à :

```
Do While Not (IsEmpty(ActiveCell))

    ActiveCell.Offset(1, 0).Activate
Loop
```

Si une cellule a un contenu :
IsEmpty(ActiveCell)=Faux
Not(Faux)=Vrai
Le While se poursuit…

Ce qui nous donnerait à peu près :

```
Option Explicit

Dim Sum As Integer
Dim rep As Integer

Sub Somme_Partielle()
   Sum = 0

   Do While Not (IsEmpty(ActiveCell))
      Sum = Sum + ActiveCell.Value
      ActiveCell.Offset(1, 0).Activate
   Loop
   ActiveCell.FormulaR1C1 = "TOTAL : "
   ActiveCell.Offset(1, 0).Activate
   ActiveCell.FormulaR1C1 = Sum
End Sub
```

Ici on se trouve sur la 1° cellule vide

Il serait facile de « professionnaliser » cette macro en demandant confirmation à l'utilisateur, en effet il y a toujours le risque d'un déclenchement intempestif

```
Sub Somme_Partielle_confirme()
   Sum = 0
    rep = MsgBox("Voulez vous réellement obtenir la somme ces
cellules non vides ?", _
                  vbOKCancel, "Macro Somme partielle")
   If rep = vbOK Then
      Do While Not (IsEmpty(ActiveCell))
         Sum = Sum + ActiveCell.Value
         ActiveCell.Offset(1, 0).Activate
      Loop
      ActiveCell.FormulaR1C1 = "TOTAL : "
      ActiveCell.Offset(1, 0).Activate
      ActiveCell.FormulaR1C1 = Sum
   Else
      rep = MsgBox("macro non éxécutée", _
                  vbOKOnly, "Macro Somme partielle")
   End If
End Sub
```

La valeur renvoyée par Message doit être récupérée mais pas forcément utilisée !

On peut noter que de toute façon, même si le nombre de cellules était connu, on ne pourrait utiliser cette fonction somme() comme dans une feuille Excel, car nous sommes dans un « module » vba

ATTENTION: le code se trouve dans le classeur **macro vba us v8 cours .xls** dans le projet **vbaxlsv8cours**, dans un module nommé **Tant_Que**

DIALOGUE UTILISATEUR

Objectif et analyse

Il s'agit de pouvoir récupérer des informations tapées par l'utilisateur lors du déroulement de la macro essentiellement sous deux formes :

- Validation de choix prédéfinis proposés dans une boite de dialogue (oui, non, abandon... etc)
  Ce dialogue repose sur la fonction **msgbox** que l'on connait
- Récupération de valeurs saisie au clavier pour effectuer des actions dans la macro
  Ce dialogue repose sur une nouvelle fonction **InputBox** et une nouvelle méthode **InputBox**

Eléments présentés :

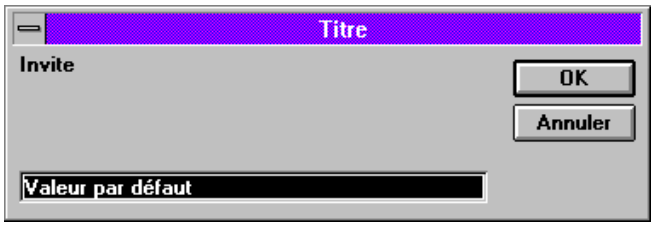
- fonction **InputBox** de visual basic
- méthode **InputBox** de Application Excel
- méthode **Address** sur objet range
- propriété **ReferenceStyle** sur objet Application

Fonction InputBox :

Il faut permettre à l'utilisateur de saisir des informations de natures diverses. C'est donc la fonction de base pour tout ce qui concerne la saisie. Cette saisie se fera par l'intermédiaire d'une boîte de dialogue à l'écran.

les paramètres principaux sont les suivant

```
Var = InputBox(Invite,Titre,Valeur par défaut)
```



Exemple :

```
Var = InputBox("Veuillez saisir le nombre d'années","Durée",15)
```

Recueillir et afficher l'information saisie :

Ci dessous une variable **rep** est utilisée pour recueillir les informations saisies par l'intermédiaire de la boîte de saisie.

```
rep = InputBox("Veuillez saisir les années", "Durée", 15)
Range("A1").Activate
ActiveCell.FormulaR1C1 = rep
```

Ci dessous le résultat de la saisie de la boîte de saisie est directement affecté à la cellule active.

```
Range("A1").Activate
ActiveCell.FormulaR1C1 = InputBox("Veuillez saisir les années", "Durée", 15)
```

Dans les 2 exemples précédents le résultat de la saisie sera affiché dans la cellule A1.

Voir exercice "Dialogue Simple" dans le support TP

Voir exercice "Dialogue Contrôlé" dans le support TP

Voir exercice "Saisie Paramètres dans Graphique" dans le support TP

Méthode InputBox :

La **méthode InputBox** est différente de la fonction **InputBox**, car elle permet de valider de façon sélective les entrées de l'utilisateur, et de recueillir des objets Excel, (notamment des objets range)

Attention :

Application.InputBox	appelle la **méthode InputBox** ;
InputBox	sans objet appelle la **fonction InputBox**.

les paramètres principaux sont quasiment identiques à ceux de la fonction, seule l'appel en **application.** diffère ainsi que le dernier paramètre **type**

```
application.InputBox(Invite, Titre, Default, Left, Top, HFile, HCont, Type)
```

Si **type** est omis, on retrouve à peu de chose près le fonctionnement de la fonction InputBox, ce qui n'est pas du tout le but…:

Si **type** est précisé, on a le fonctionnement de la fonction InputBox plus :

- la possibilité de contrôler le type de l'entrée
- la possibilité de cliquer une cellule voire une plage…

Contrôler le type d'entrée

Type = 1 : Récupération d'une valeur numérique :

```
Sub Saisie_Nombre()
   Dim Repnb As Integer
   Repnb = 69
   Repnb = Application.InputBox("saisir nb ", "bonjours", 4, , , , , 1)
   rep = MsgBox(Repnb, vbOKOnly, "vous venez de taper")
   ActiveCell().Value = Repnb
   ActiveCell.Offset(1, 0).Activate
End Sub
```

Essayer de saisir :

- un entier
- du texte
- un décimal

La simple modification du type de variable permet la récupération d'un nombre à virgule

```
Dim Repnb As Single
```

Type = 2 : Récupération d'une valeur texte :

```
Sub Saisie_Texte()
   Dim Repst As String
   Repst = Application.InputBox("saisir texte", "bonjours", "youpee", , , , , 2)
   rep = MsgBox(Repst, vbOKOnly, "vous venez de taper")
   ActiveCell().Value = Repst
   ActiveCell.Offset(1, 0).Activate
End Sub
```

Type = 0 : Récupération d'une formule :

```
Sub Saisie_Formule()
   Dim Repchaine As String
      Repchaine = Application.InputBox("saisir form", "bonjours", "=", , , , , 0)
      rep = MsgBox(Repchaine, vbOKOnly, "vous venez de taper")
      ActiveCell().Value = Repchaine
      ActiveCell.Offset(1, 0).Activate
End Sub
```

Essayer de saisir :

- **=somme(**
- sélectionner les cellules voulues
- taper **)** et valider

Type = 8 : Récupération d'une plage :

dans l'écriture

```
Dim rep as range
```

on déclare une variable de type range, et on lui donne ensuite la référence saisie dans la boite de dialogue via l'instruction **set** comme dans

```
set rep=application.InputBox(……..,8)
```

pour obtenir l'adresse d'une plage il faut utiliser la méthode address comme dans

```
rep.adress()
Sub Saisie_Plage()
```

ce qui pourrait donner le code suivant

```
Sub Saisie_cel
   Dim RepR As Range
      Set RepR = Application.InputBox("saisir plage", "bonjours", "A1:A8", , , , ,
8)
      Rep = MsgBox(Rep.Address(), 0, "vous venez de sélectionner")
      ActiveCell.Offset(1, 0).Activate
      RepR.Select
      ActiveCell.Offset(1, 0).Activate
End Sub
```

Méthode Address :

La méthode **Address** renvoie la référence de la Plage, sous forme d'une chaîne de caractères.

références A1 ou Lxcy

on peut facilement savoir si Excel est en notation A1 ou L1C1

```
If Application.ReferenceStyle = xlR1C1 Then
   MsgBox ("Microsoft Excel utilise les références L1C1")
Else
   MsgBox ("Microsoft Excel utilise les références A1")
End If
```

et même facilement le modifier à la volée par

```
Application.ReferenceStyle = xlA1                    ' ou xlR1C1
```

en conséquence de quoi il paraît intéressant de pouvoir afficher les adresses de manière conforme à l'environnement :

étant positionné sur la cellule A2

```
Range("A2").Activate
   Set rep = ActiveCell
```

on affiche ses coordonnées par défaut

```
   MsgBox rep.Address()                    ' $A$2
```

on affiche ses coordonnées en indiquant la colonne en absolu

```
   MsgBox rep.Address(RowAbsolute:=False)          ' $A2
```

on affiche ses coordonnées en notation L1C1

```
   MsgBox rep.Address(ReferenceStyle:=xlR1C1)       ' L2C1
```

on affiche ses coordonnées en notation L1C1 en relatif …

```
MsgBox rep.Address(ReferenceStyle:=xlR1C1, _
   RowAbsolute:=False, _
   ColumnAbsolute:=False)                                ' L(-2)C(-2)
```

par rapport à cells(3,3) …

```
   MsgBox rep.Address(ReferenceStyle:=xlR1C1, _
   RowAbsolute:=False, _
   ColumnAbsolute:=False, _
```

ATTENTION: le code se trouve dans le classeur **macro vba us v8 cours .xls** dans le projet **vbaxlsv8cours**, dans un module nommé **Dialogue_Simple**

Voir exercice "Somme et Positionnement" dans le support TP

SELECTION DE CELLULE

Objectif :

Pouvoir sélectionner une cellule ou une plage de cellules avec des coordonnées absolues ou relatives, par rapport à la cellule de départ (celluleactive) ou décalées par rapport à la cellule de départ

Elements présentés:

- Méthode **Select** sur objet range
- Méthode **Union** sur objet range
- Propriété **Offset** sur objet range

Adressage Absolu :

Pour sélectionner de manière absolue, deux moyens principaux existent

- propriété **Range**
- Propriété **Cells**

à l'aide de la propriété **Range** :,

sélection d'une cellule

pour sélectionner une cellule on peut écrire

```
Range("B3").Select
```

comme la cellule active est par défaut sélectionner parfois on vois

```
Range("B3").Activate
```

on peut faire référence à plusieurs plages plaçant des virgules entre deux ou plusieurs références

sélection d'une plage

pour sélectionner une plage de cellule on peut écrire

```
Range("B3:D3").Select
```

On peut écrire aussi

```
Range("B3:D3,H2,C8:C12").Select
```

à l'aide de la propriété **Cells** déjà étudiée, on peut écrire

```
Cells(3, 2).Select
```

ou à la rigueur

```
Range(Cells(3, 2), Cells(3, 4)).Select
```

Adressage nommé :

Les plages nommées permettent à la propriété **Range** de travailler plus facilement sur plusieurs plages. L'exemple suivant fonctionne si les trois plages nommées se trouvent sur la même feuille.

```
Sub ClearNamed()
    Range("maPlage, taPlage, saPlage").ClearContents
End Sub
```

Méthode Union :

Vous pouvez combiner plusieurs plages en un seul objet Range à l'aide de la méthode Union.

L'exemple suivant montre comment créer un objet Range nommé plusieursPlages, le définir comme étant constitué des plages A1:B2 et C3:D4 puis met l'ensemble en gras.

```
Sub MultipleRange()
    Dim r1, r2, plusieursPlages As Range
    Set r1 = Sheets("Feuil1").Range("A1:B2")
    Set r2 = Sheets("Feuil1").Range("C3:D4")
    Set plusieursPlages = Union(r1, r2)
    plusieursPlages.Font.Bold = True
End Sub
```

Adressage Relatif :

Pour sélectionner de manière relative, deux moyens principaux existent

- propriété **Range**
- Propriété **Cells**

à l'aide de la propriété **Range** déjà étudiée,

sélection d'une cellule

pour sélectionner une cellule on peut écrire

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

sélection d'une plage

on peut faire référence à plusieurs plages plaçant des : entre deux ou plusieurs références

```
ActiveCell.Range("A1:C1").Select
```

N.B: Les références n'expriment que la géométrie de la plage, car la position de cette plage ne dépends que de la cellule Active.

ainsi :

```
ActiveCell.Range("A1:E1").Select
```

Sélectionne 5 cellules en ligne à partir de la cellule active

```
ActiveCell.Range("A1:A10").Select
```

Sélectionne 10 cellules en colonne à partir de la cellule active

```
ActiveCell.Range("A1:E10").Select
```

Sélectionne 50 cellules (10 colonnes / 5 lignes) à partir de la cellule active

à l'aide de la propriété **Cells** déjà étudiée,

sélection d'une cellule

pour sélectionner une cellule on peut écrire

```
ActiveCell.Select
```

sélection d'une plage

pour sélectionner une plage on peut écrire

```
ActiveCell.Range(Cells(1,1): Cells(1,3)).Select
```

N.B: Les références n'expriment que la géométrie de la plage, car la position de cette plage ne dépends que de la cellule Active.

ainsi :

```
ActiveCell.Range(Cells(1,1): Cells(1,5)).Select
```

Sélectionne 5 cellules en ligne à partir de la cellule active

```
ActiveCell.Range(Cells(1,1): Cells(10,1)).Select
```

Sélectionne 10 cellules en colonne à partir de la cellule active

```
ActiveCell.Range(Cells(1,1): Cells(5,10)).Select
```

Sélectionne 50 cellules (10 colonnes / 5 lignes) à partir de la cellule active

---

Adressage Relatif Décalé :

Pour sélectionner de manière relative et décalée par rapport à la cellule active, un moyen existe

- propriété **Offset**

```
ActiveCell.Offset(nbre ligne, nbre colonne).Range("A1").Select
```

N.B: Les références n'expriment que la géométrie de la plage, car la position de cette plage ne dépends que de la cellule Active+Offset.

ainsi :

```
ActiveCell.Offset(1, 0).Range("A1").Select
```

sélectionne une cellule décalée d'une ligne vers le bas

```
ActiveCell.Offset(0, 1).Range("A1").Select
```

sélectionne une cellule décalée d'une colonne vers la droite

```
ActiveCell.Offset(-1, 0).Range("A1").Select
```

sélectionne une cellule décalée d'une ligne vers le haut

```
ActiveCell.Offset(0, -1).Range("A1").Select
```

sélectionne une cellule décalée d'une colonne vers la gauche

```
ActiveCell.Offset(1, 1).Range("A1").Select
```

sélectionne une cellule décalée d'une colonne vers la droite et d'une ligne vers le bas

et on pourrait aussi écrire :

```
ActiveCell.Offset(1, 1).Range("A1").Select
```

notation abrégée car on ne sélectionne qu'une cellule

```
ActiveCell.Offset(1,0).Range("A1:E1").Select
```

sélectionne 5 cellules en ligne décalée d'une ligne vers le bas par rapport à la cellule active…

etc…

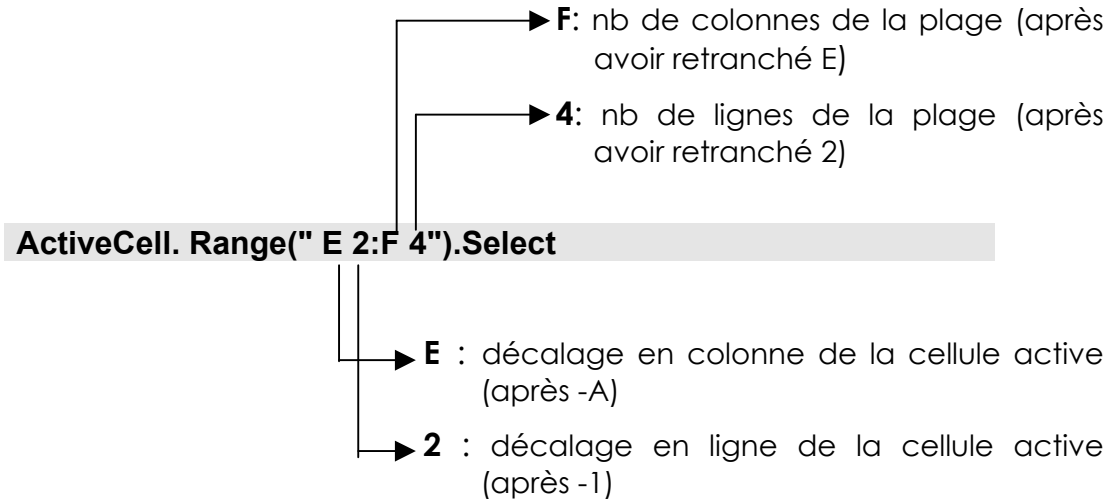
Remarque :

Comme vous pouvez le constater l'instruction ***ActiveCell.Offset(x,y)***… est surtout intéressante lorsque l'on veut se décaler d'une position initiale. Lorsque l'on voudra créer une procédure pouvant intervenir n'importe où dans une feuille de calculs, il suffira tout simplement de positionner le curseur de travail avant de lancer l'enregistreur de macro.

Adressage Relatif Décalé notation abrégée :

jusqu'ici toutes les notations commencent en "A1", c'est à dire sur la celluleActive qui est prise comme point de départ de référence. on peut commencer ailleurs, et le décalage s'obtient par la différence entre A1 et la première coordonnée écrite dans Range("...:...").

Ainsi dans l'exemple :



Ainsi

```
ActiveCell.Offset(1,0).Range("A1:A3").Select
```

peut s'écrire:

```
ActiveCell.Range("A2:A4").Select
```

et

```
ActiveCell.Offset(2,4).Range ("A1:D3").Select
```

peut s'écrire:

```
ActiveCell.Range ("E2:H4").Select
```

N.B: Si cette écriture est relativement incompréhensible, elle est parfois générée par l'enregistreur de macro ! !

SÉLECTIONS PARTICULIERES

Sélectionner des cellules de façon rapide. Equivalent de quelques combinaisons de touches

Elements présentés:

- Méthode **Select** sur objet range
- Méthode **Union** sur objet range
- Propriété **CurrentRegion** sur objet range
- Propriété **EntireColumn** sur objet range
- Propriété **EntireRow** sur objet range

Sélection de la zone en cours CTRL+*:

Sélectionne toutes les cellules aux alentours de la cellule active

Equivaut à la sélection Ctrl * ou Equivaut au bouton 

SOCIETES	PAYS	CH-AFF 90	CH-AFF 91	CROISSANCE
Framtour	Fr	47 812 457.00	50 000 000.00	-4.58%
Klups	All	4 500 000.00	4 700 000.00	-4.44%
Rhin-Boulac	Fr	9 542 167.00	10 000 000.00	-4.80%
Scheind	All	4 624 897.00	4 500 000.00	2.70%
Wolks	All	10 005 000.00	11 000 000.00	-9.95%

⇩

SOCIETES	PAYS	CH-AFF 90	CH-AFF 91	CROISSANCE
Framtour	Fr	47 812 457.00	50 000 000.00	-4.58%
Klups	All	4 500 000.00	4 700 000.00	-4.44%
Rhin-Boulac	Fr	9 542 167.00	10 000 000.00	-4.80%
Scheind	All	4 624 897.00	4 500 000.00	2.70%
Wolks	All	10 005 000.00	11 000 000.00	-9.95%

On peut écrire

```
Selection.CurrentRegion.Select
```

ou

```
ActiveCell.CurrentRegion.Select
```

et donc pour mettre le tableau en style monétaire

```
Sub Region()
    ActiveCell.CurrentRegion.Select
    Selection.Style = "Monétaire"
End Sub
```

Sélection du coin supérieur gauche d'une zone en cours

Objectif :

Lorsque vous avez effectué une sélection d'une zone en cours vous pouvez par cette instruction sélectionner uniquement la cellule se trouvant dans le coin supérieur gauche

SOCIETES	PAYS	CH-AFF 90	CH-AFF 91	CROISSANCE
Framtour	Fr	47 812 457.00	50 000 000.00	-4.58%
Klups	All	4 500 000.00	4 700 000.00	-4.44%
Rhin-Boulac	Fr	9 542 167.00	10 000 000.00	-4.80%
Scheind	All	4 624 897.00	4 500 000.00	2.70%
Wolks	All	10 005 000.00	11 000 000.00	-9.95%

SOCIETES	PAYS	CH-AFF 90	CH-AFF 91	CROISSANCE
Framtour	Fr	47 812 457.00	50 000 000.00	-4.58%
Klups	All	4 500 000.00	4 700 000.00	-4.44%
Rhin-Boulac	Fr	9 542 167.00	10 000 000.00	-4.80%
Scheind	All	4 624 897.00	4 500 000.00	2.70%
Wolks	All	10 005 000.00	11 000 000.00	-9.95%

Syntaxe :

```
ActiveCell.select
```

Voir exercice "Séléction Plage dans graphique" dans le support TP

Sélection d'une colonne complète

Objectif :

Lorsque vous avez effectué une sélection d'une colonne complète, on peut facilement l'effacer ou la supprimer

Syntaxe :

ActiveCell.EntireColumn.	suivit de l'action

ainsi

ActiveCell.EntireColumn.Delete	supprime la colonne courante

et

ActiveCell.EntireColumn.ClearContents	efface le contenu de la colonne

Sélection d'une ligne complète

Objectif :

Lorsque vous avez effectué une sélection d'une ligne complète, on peut facilement l'effacer ou la supprimer

Syntaxe :

ActiveCell.EntireRow.	suivit de l'action

ainsi

ActiveCell.EntireRow.Delete	supprime la ligne courante

et

ActiveCell.EntireRow.ClearContents	efface le contenu de la ligne

GESTION DE FEUILLE

objectif :

Nommer une feuille du classeur à partir d'une réponse de l'utilisateur ou de donnée internes, qu'il s'agisse de variables ou de paramètres systèmes (horodatage, chemin...)

Etre capable de gérer les feuilles d'un classeur, en les supprimant, créant…

Nommer une feuille :

Il faut se positionner sur la feuille à nommer (si on n'y est pas déjà) , avec une instruction du genre

```
ActiveWorkbook.Worksheets(1).Activate
```

puis pour la nommer écrire une instruction du genre :

```
ActiveSheet.Name = "nom à donner"
```

On utilise ici

- Propriété **Name** sur objet **Worksheet**

voilà une macro qui nomme la feuille d'un classeur à partir du contenu de la cellule active au moment de l'exécution du code, puis qui se positionne sur la première feuille du classeur

```
Sub Nomme_Feuille()
   ActiveSheet.Name = ActiveCell.Value

   ActiveWorkbook.Worksheets(1).Activate

End Sub
```

Voir exercice "Nommer une feuille" dans le support TP

Ajouter une feuille :

Il suffit d'utiliser la méthode **Add** sur une collection de type **Worksheets** avec une instruction du genre

```
ActiveWorkbook.Worksheets.Add
```

On utilise ici

- méthode **Add** sur objet (collection) **Worksheets**

voilà une macro qui crée une nouvelle feuille dans un et la nomme automatiquement avec la date du jour au format jj-mm-aa

```
Sub Ajout_Feuille()
   ActiveWorkbook.Worksheets.Add
   ActiveSheet.Name = Day(Date) & "-" & Month(Date) & "-" & Year(Date)
End Sub
```

Supprimer une feuille :

On utilise ici

- Méthode **Delete** sur objet **Worksheet (ou collection Worksheets)**

Pour supprimer la feuille courante il suffit de dire

```
ActiveSheet.Delete
```

Voilà une macro qui supprime la feuille nommée "toto"

```
Sub Supp_Feuille()
  ActiveWorkbook.Worksheets("toto").Delete
End Sub
```

On a remarqué que Excel nous demande confirmation, ce qui est normal.

mais on peut demander que en cours d'exécution d'une macro, Excel ne demande pas de confirmation, cela s'obtient par une propriété de l'objet Application, qui peut être mise à false

On utilise alors

- Propriété **DisplayAlerts** sur objet **Application**

ainsi écrire

```
Application.DisplayAlerts = False
```

empêche à Excel de demander confirmation pour la suppression d'un feuille par exemple…

Voilà une macro qui crée une nouvelle feuille nommée vec la date du jour, puis, après validation de la part de l'utilisateur la supprime !

```
Sub Ajout_Supp_Feuille()
   ActiveWorkbook.Worksheets.Add
   ActiveSheet.Name = Day(Date) & "-" & Month(Date) & "-" &
Year(Date)
   MsgBox "cliquez Ok pour continuer"
   ActiveSheet.Delete
End Sub
```

Voir exercice "Ajouter - Supprimer une feuille" dans le support TP

Parcourir toutes les feuilles d'un classeur :

On peut avoir besoin de parcourir toutes les feuilles d'un classeur, et à terme s'y déplacer…

On peut utiliser ici deux techniques , une "classique", dans laquelle on récupère le nombre de feuille et on se promène ensuite avec une boucle, ou une méthode un peu plus "objet" dans laquelle on demande de parcourir tous les éléments de l'objet par une instruction appropriée…

méthode "classique" :

on a besoin d'utiliser

- Propriété **Count** sur objet collection **Worksheets**

Qui renvoie le nombre d'éléments de la collection

```
Sub nb_feuille()
Dim nbfeuilles As Integer

   nbfeuilles = Worksheets.Count
   MsgBox "il existe " & nbfeuilles & " feuilles de calcul dans ce classeur"
End Sub
```

et puis ensuite on parcoure avec une boucle **for**

```
Sub Parcoure_feuille()
Dim i, nb As Integer
Dim feuille As Worksheet

   nb = Worksheets.Count
   For i = 1 To nb Step 1
      Worksheets(i).Activate
      MsgBox " on se promène sur la" & i & " feuille "
   Next

End Sub
```

méthode "objet" :

il nous faut déclarer une variable du type de l'objet que l'on va parcourir, donc ici un worksheet

```
Dim feuille As Worksheet
```

puis utiliser l'instruction **For each**

```
Sub Parcoure_feuille2()
Dim rep, i As Integer
Dim feuille As Worksheet

   For Each feuille In Worksheets
      feuille.Activate
      MsgBox " on se promène sur la feuille " & feuille.Name
   Next feuille

End Sub
```

Déplacer une feuille d'un classeur :

On peut avoir besoin de déplacer une feuille…

on a besoin d'utiliser

- Méthode **Move** sur objet **Worksheet**

par exemple cette macro déplace la feuille active après la dernière feuille du classeur :

```
Sub Deplace_feuille()
Dim nbfeuilles As Integer

   nbfeuilles = Worksheets.Count
   ActiveSheet.Move After:=Worksheets(nbfeuilles)

End Sub
```

Copier une feuille d'un classeur :

On peut avoir besoin de copier une feuille…

on à besoin d'utiliser

- Méthode **Copy** sur objet **Worksheet**

par exemple cette macro copie la feuille active après la dernière feuille du classeur :

```
Sub Copie_feuille()
Dim nbfeuilles As Integer

   nbfeuilles = Worksheets.Count
   ActiveSheet.Copy After:=Worksheets(nbfeuilles)

End Sub
```

ATTENTION: le code se trouve dans le classeur **macro vba us v8 cours .xls** dans le projet **vbaxlsv8cours**, dans un module nommé **Gestion_de_Feuille**

Voir exercice "Créer manipuler des feuilles" dans le support TP

GESTION DE CLASSEUR

Objectif :

Connaître le nom d'un classeur, son endroit de stockage.

Ouvrir, un classeur Excel, et si possible être capable de s'assurer au préalable de son existence.

Nom d'un classeur et Chemin :

pour afficher le nom d'un classeur, cela ne pose pas de problème (la propriété **name** fait très bien l'affaire) mais pour connaître l'endroit où il est stocké, on a besoin d'utiliser

- Propriété **Path** sur objet **Workbook**

par exemple cette macro affiche le nom est le chemin dans lequel le classeur courant est enregistré

```
Sub Info_Classeur()
   rep = MsgBox("vous êtes dans le fichier " & ActiveWorkbook.Name)
   rep = MsgBox("fichier situé en " & ActiveWorkbook.Path)
End Sub
```

N.B: si le classeur n'est pas enregistré, la chaine vide "" est renvoyée par **Path**

Ouvrir un classeur:

pour ouvrir un classeur, sans précaution particulières (on suppose que celui-ci existe) on a besoin d'utiliser

- Méthode **Open** sur objet (collection) **Workbooks**

```
Sub Ouverture_simple()
   Libelle = "Nom du fichier à ouvrir"
   Title = "OUVERTURE FICHIER"
   NomFich = Application.InputBox(Libelle, Title, "bidon.xls", , , , , 2)

   Workbooks.Open Filename:=NomFich

End Sub
```

Mais on peut aussi être plus prudent, et avant d'ouvrir un classeur vérifier que celui-ci existe. On a alors besoin d'utiliser

- Fonction **Dir**

Ainsi dans cette macro, après avoir demandé le fichier à ouvrir à l'utilisateur, on vérifie son existence, avant de lancer la commande d'ouverture

```
Sub Ouverture_Verif()
   Dim NomFich, Nomch As String

   Libelle = "Nom du fichier à ouvrir"
   Title = "OUVERTURE FICHIER"
   NomFich = Application.InputBox(Libelle, Title, , , , , 2)

   Nomch = Dir(NomFich)
   'MsgBox (Nomch)
   If (Nomch <> "") Then
      Workbooks.Open Filename:=NomFich
   Else
      rep = MsgBox("desolé fichier inexistant !", vbOKOnly, "attention")
   End If
End Sub
```

Enregistrer un classeur :

Pour enregistrer un classeur, sans précaution particulières (on suppose que celui-ci existe, sinon on l'enregistre avec son nom par défaut classeur1.xls dans le dossier par défaut…) on a besoin d'utiliser

- Méthode **Save** sur objet **Workbook**

Ainsi ici on enregistre la classeur actif

```
Sub Enreg()

   ActiveWorkbook.Save
End Sub
```

Enregistrer un classeur (sous) :

Pour enregistrer un classeur avec son nom on a besoin d'utiliser

- Méthode **SaveAs** sur objet **Workbook**

Ici on demande à l'utilisateur le nom sous lequel on va enregistrer le classeur

```
Sub Enreg_sous()
   Libelle = "Nom du fichier à enregistrer"
   Title = "CREATION DE FICHIER"
   NomFich = Application.InputBox(Libelle, Title, , , , , 2)
   ActiveWorkbook.SaveAs NomFich
                  'ActiveWorkbook.SaveAs Filename:=NomFich
End Sub
```

Fermer un classeur :

Pour fermer un classeur on a besoin d'utiliser

- Méthode **Close** sur objet **Workbook**

Ici on demande de fermer le classeur actif sans enregistrer les modifications

```
Sub fermer_ssmodif()

 ActiveWorkbook.Close SaveChanges:=False
End Sub
```

ATTENTION: le code se trouve dans le classeur **macro vba us v8 cours .xls** dans le projet **vbaxlsv8cours**, dans un module nommé **Gestion_de_Classeur**

APPEL DE FONCTION

Objectif :

Utiliser une fonction dans une procédure, pour simplifier la lecture et les modifications, c'est à dire structurer le code

Eléments présentés :

- Notion de Fonction / Procédure

Codes exemples :

Voilà une déclaration de fonction de type bouléen, et par conséquent utilisable partout où ce type de valeur peut être utilisé, notamment dans une comparaison

```
Function ExisteFichier(NomFich) As Boolean
   ExisteFichier = (Dir(NomFich) <> "")
End Function
```

Utilisation de cette fonction comme « primitive » :

```
Sub Ouvre_Fichier()
   Libelle = "Nom du fichier à ouvrir"
   Title = "OUVERTURE FICHIER"
   NomFich = Application.InputBox(Libelle, Title, , , , , , 2)

   If ExisteFichier(NomFich) Then
      Workbooks.Open Filename:=NomFich
   Else
      MsgBox "desolé fichier inexistant !", vbOKOnly, "attention"
   End If

End Sub
```

ATTENTION: le code se trouve dans le classeur **macro vba us v8 cours .xls** dans le projet **vbaxlsv8cours**, dans un module nommé **Appel_Fonction**

IMPRESSION

Imprimer par défaut :

Pour imprimer une feuille en utilisant la détection automatique de la zone à imprimer par Excel on à besoin de

- Méthode **PrintOut** sur objet **Worksheet**

Ainsi cette macro Imprimer une feuille (zone par défaut) mais demande auparavant une pré-visualisation

```
Sub imprime_défaut()

   ActiveSheet.PrintOut , , , True
End Sub
```

False imprimerait directement

Imprimer une zone définie :

Pour imprimer une feuille en spécifiant la zone à imprimer par Excel on à besoin de

- Propriété **PageSetup** sur objet **Worksheet**

Ainsi cette macro Imprime directement les cellules A1:A7 d'une feuille

```
Sub imprime_abs()

   ActiveSheet.PageSetup.PrintArea = "$A$1:$C$7"
   ActiveSheet.PrintOut , , , False
End Sub
```

Imprimer une zone sélectionnée :

L'utilisateur sélectionne la zone à imprimer, puis déclenche la macro

```
Sub imprime_rel()
   ActiveCell.Range("A1:C7").Select
   ActiveSheet.PageSetup.PrintArea = Selection.Address
   ActiveSheet.PrintOut , , , True
End Sub
```

Imprimer une zone sélectionnée à la demande:

L'utilisateur déclenche la macro, qui demande alors de sélectionner la zone à imprimer

```
Sub imprime_zone_choisie()
Dim zone As Range
   Set zone = Application.InputBox("selectionnez la zone à imprimer",
"attention", , , , , , 8)
   zone.Select
   ActiveSheet.PageSetup.PrintArea = Selection.Address
   ActiveSheet.PrintOut , , , True
End Sub
```

Cette procédure peut être améliorée en « proposant comme zone d'impression la zone en cours (CTRL *) autour de la cellule active, ce que l'on obtient par l'ajout de

```
ActiveCell.CurrentRegion.Address(ReferenceStyle:=xlR1C1)
```

dans le paramètre « par défaut » dans la boite de dialogue InputBox

ce qui donnerait finalement

```
Sub imprime_zone_choisie2()
Dim zone As Range

   Set zone = Application.InputBox("selectionnez la zone à imprimer", "attention",
      ActiveCell.CurrentRegion.Address(ReferenceStyle:=xlR1C1), , , , , 8)
   zone.Select
   ActiveSheet.PageSetup.PrintArea = Selection.Address
   ActiveSheet.PrintOut , , , True
End Sub
```

ATTENTION: le code se trouve dans le classeur **macro vba us v8 cours .xls** dans le projet **vbaxlsv8cours**, dans un module nommé **Imprime**

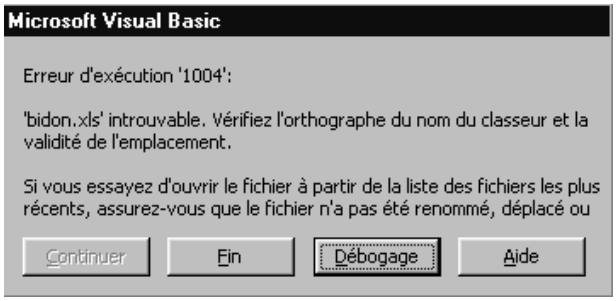
DEBUGGER

Erreurs de Compilation, d'Execution, de Logique :

Lorsqu'une erreur survient, on peut généralement la classer dans une des trois catégories suivantes :

Compilation : facilement détectables, parfois même par l'analyseur de syntaxe, ce sont les erreurs les plus faciles à corriger

Exécution : Une macro est interrompue parce que l'exécution d'une ligne de code est impossible, comme l'ouverture d'un fichier inexistant, une division par 0…On peut y remédier « a la volée » si l'analyse d'Excel est assez pertinente



Logique : Ce sont les erreurs les plus difficiles à détecter car la macro fonctionne mais ne donne pas le résultat escompté. A ce niveau le débugger peut être indispensable pour essayer de comprendre

Le Débugger :

La fenêtre du Debugger s'affiche depuis un module par le Menu :

Affichage / fenêtre Execution ou la combinaison **CTRL+G**

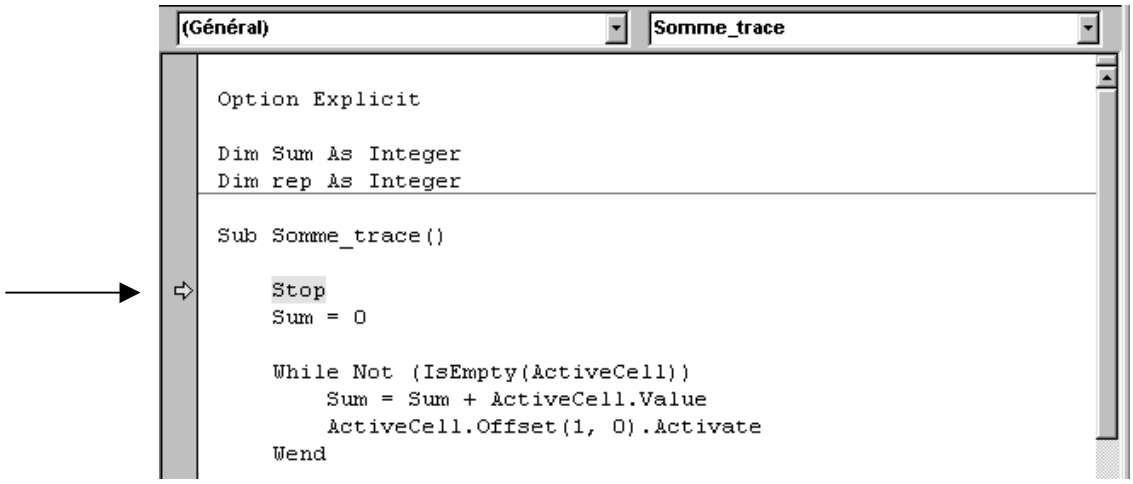
Mais on peut la faire apparaître par l'instruction **STOP** dans le code

Essayons de visualiser le fonctionnement de notre fonction de somme partielle « itérative »

```
Sub Somme_trace()
   Stop
   Sum = 0
   While Not (IsEmpty(ActiveCell))
      Sum = Sum + ActiveCell.Value
      ActiveCell.Offset(1, 0).Activate
   Wend
   ActiveCell.FormulaR1C1 = "TOTAL : "
   ActiveCell.Offset(1, 0).Activate
   ActiveCell.FormulaR1C1 = Sum
End Sub
```

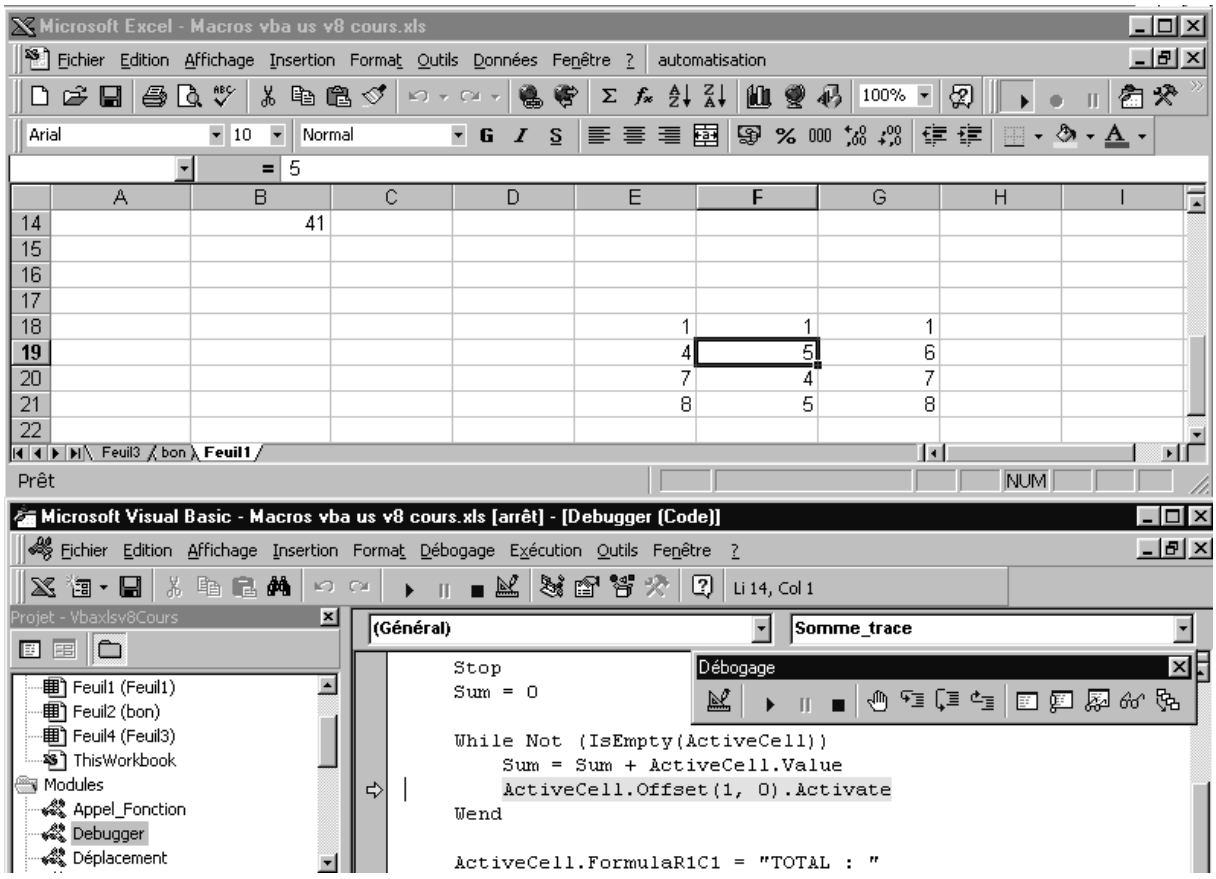
Mode Arrêt via Débugger

L'éxecution de cette macro provoque l'affichage du débugger, a l'intérieur de la fenêtre de développement Visual basic, comme ci-dessous



On en profite pour afficher la barre d'outils du Débogage

Il faut alors réduire la fenêtre de l'éditeur Visual basic pour pouvoir observer également "ce qui se passe" dans la fenêtre Excel,



Le pas à pas :

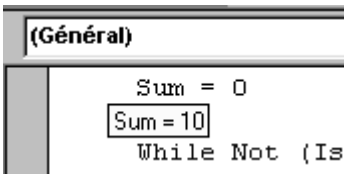
Il est possible d'avancer dans la programme en utilisant les 3 outils :

- **Pas à Pas détaillé** (F8)
- **Pas à Pas principal**
- **Pas à Pas Sortant**

ou reprendre le déroulement normal avec l'outils **Continuer** (F5) 

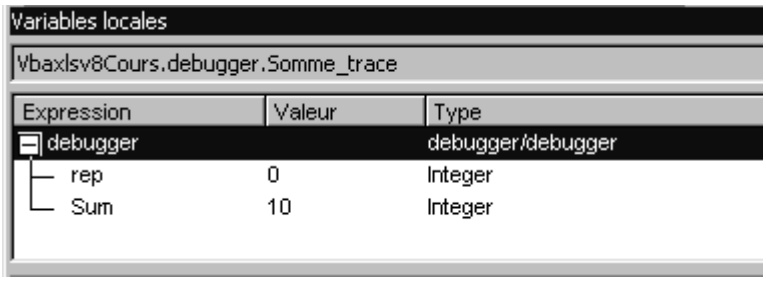
connaître la valeur des variables :

normalement, le simple fait de "passer" sur une variable avec la souris, permet d'afficher sa valeur dans un e info_bulle



et la fenêtre variable locale → 

affiche elle toutes les variables locales



Intervenir en cours d'execution :

cela peut se faire par l'intermediaire d'une fenêtre execution → 

dans laquelle il suffit de taper l'instruction que l'on souhaite effectuer

De manière générale vous pouvez dans le volet exécution taper toute instruction comme vous la taperiez dans le module

Point d'arrêt :

Basculer le point d'arrêt :

Permet de définir le prochain endroit où l'on souhaite que l'exécution de la macro s'arrête (après redémarrage via le bouton ) ou permet de supprimer un point d'arrêt existant

les lignes « point d'arrêt » sont mises en couleur

On peut supprimer tous les points d'arrêt par le menu :

Débogage / effacer les points d'arrêts