

Cours d'Excel approfondi

Macros et VBA



Cours d'Excel approfondi Macros et VBA 15h

Version Excel 2010 - 2013

Remarque : site permettant de répondre à beaucoup de questions sur VBA

<http://vb.developpez.com/faqvba/>

<https://openclassrooms.com/courses/825502-analysez-des-donnees-avec-excel/822888-premiers-pas-en-vba>

<https://openclassrooms.com/courses/825502-analysez-des-donnees-avec-excel/822767-les-macros>

Introduction : Quelle est l'utilité de maîtriser les macro-commandes et VBA sous XL ?

Avant tout : Pour améliorer **votre employabilité** sur le marché du travail (en complément de la maîtrise de R, Python, SAS, ...) sur des emplois ou stages de chargés d'étude économie

Exemple : aller sur le site de recherche d'emploi / stage Glassdoor

<https://www.glassdoor.fr/index.htm> et taper VBA ...

Excel : tableur qui possède de nombreuses commandes (et de manière croissante avec les différentes versions) permettant de faire énormément de choses : automatisation des calculs (c'est la base des tableurs), fonctions mathématiques, statistiques ..., graphiques, tableaux croisés dynamiques etc.

Problématique :

Les commandes prédéfinies par XL permettent donc de faire les **opérations standards** les plus utilisées par la majorité des utilisateurs.

Cependant : ces commandes ne sont **pas personnalisées** et donc peuvent se révéler insuffisantes pour réaliser des tâches très spécifiques exigées par votre travail.

Or, la **recherche de la productivité** est le fondement des logiciels de bureautiques (cf. « logiciels de productivité ») : comment gagner du temps sur des tâches **courantes** que vous devez faire de manière répétée alors qu'il n'existe pas de commandes prédéfinies par XL ?

Les solutions :

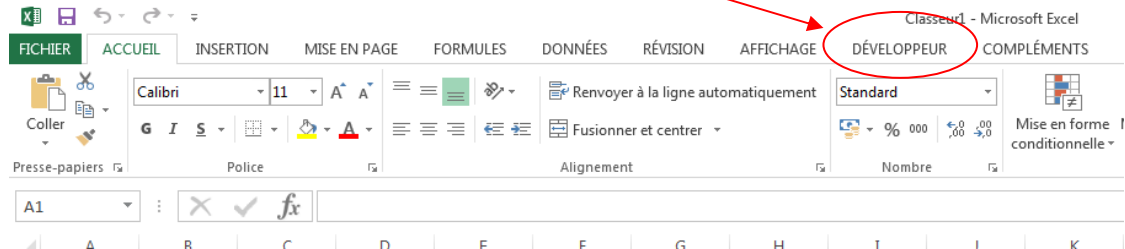
1^{er} niveau de réponse : l'utilisation de **macros**

2^{ème} niveau de réponse (beaucoup plus puissant) : la **programmation sous VBA**

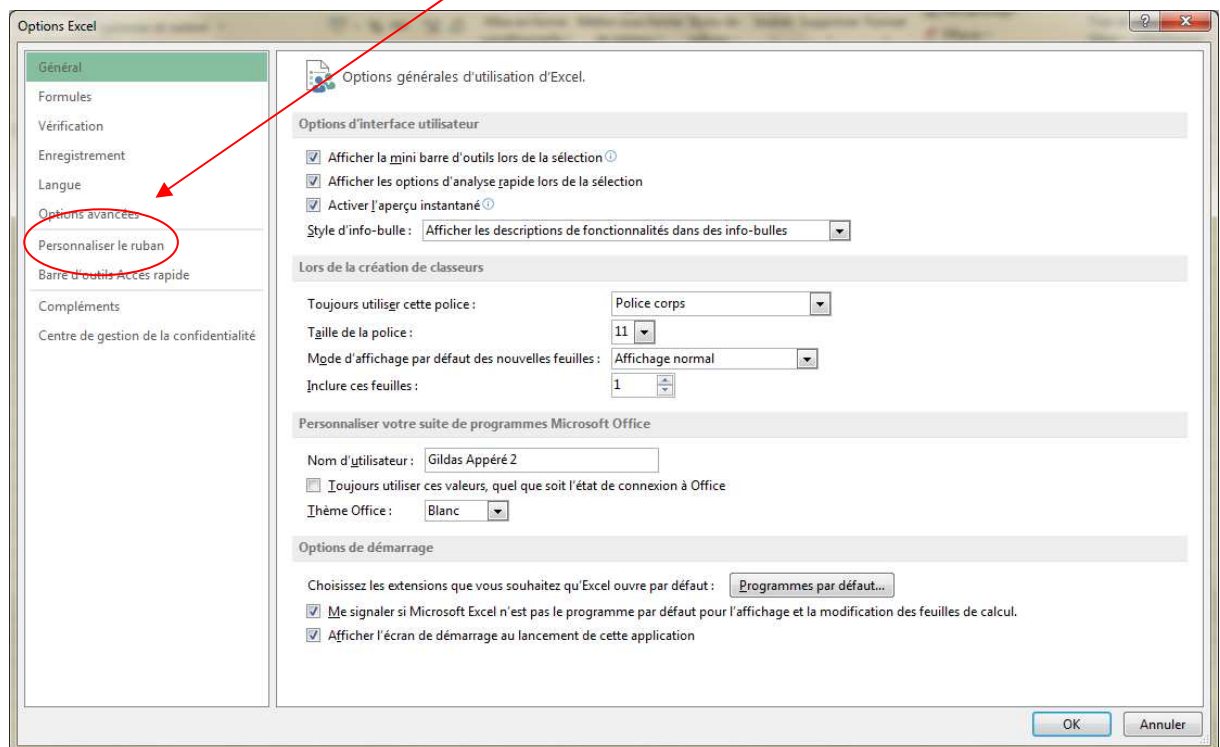
1. L'utilisation des macros sous XL

1.1. Installation de Visual Basic sous XL

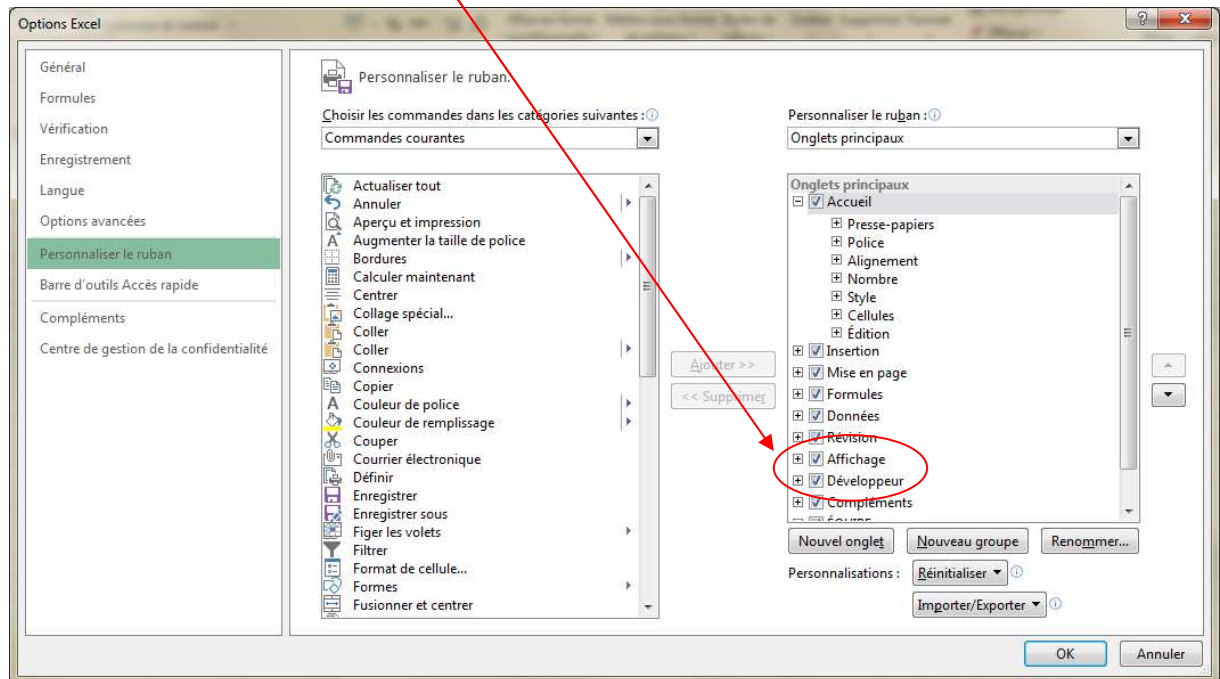
Si l'onglet **Développeur** n'apparaît pas ...



Aller dans : Fichier → Options → Personnaliser le ruban



Cocher l'onglet Developpeur



1.2. Définition et description d'une macro

Voir aussi (par exemple) : <https://openclassrooms.com/courses/825502-analysez-des-donnees-avec-excel/822767-les-macros>

Définition :

Les macros sont un moyen simple **d'automatiser des commandes** qui n'existent pas directement sous XL via un bouton dédié du fait que ces commandes sont trop spécifiques à l'utilisateur et / ou peu répandues. Une large partie de ces macros consistent à enregistrer **une combinaison** de commandes simples existant déjà sous XL telles que par exemple : mettre en gras, en italique, centrer le contenu d'une cellule ou d'une plage de cellules. Comme indiqué précédemment, le but des macros est de simplifier le travail et/ou d'augmenter la vitesse d'exécution de tâches répétitives.

Les macros sont en fait des programmes (i.e. avec des lignes de programmation) fonctionnant en Visual Basic. Même s'il n'est nullement nécessaire de recourir à VBA pour faire fonctionner une macro, savoir comment fonctionne une macro constitue une très bonne introduction à VBA (voir plus loin).

Principe général :

Le principe de base d'une macro est très simple : il s'agit **d'enregistrer une séquence** de commandes (plus ou moins longue et complexe) que l'on pourra mobiliser à volonté : en un seul clic XL réalise alors automatiquement toutes les commandes contenues dans la macro.

Par exemple la macro « Centrer_Gras_Italique » exécutera en un seul clic et simultanément la mise en forme Centrer / Mettre en Gras / Mettre en Italique sur la plage de cellules sélectionnée.

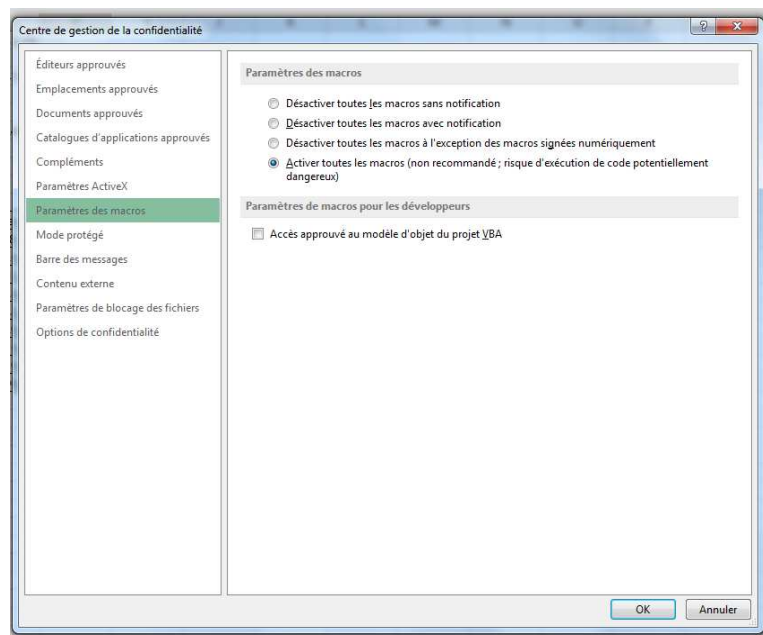
Remarques importantes :

- **Virus associés aux macros** et Niveau de sécurité d'XL

Comme l'indique l'Aide sous XL concernant les macros : « Soyez prudent lorsque vous exécutez des macros, car elles peuvent contenir des virus. Prenez les précautions suivantes : exécutez un logiciel antivirus à jour sur votre ordinateur ; choisissez un niveau de sécurité élevé pour votre macro ; désactivez la case à cocher **Faire confiance à tous les modèles et compléments installés** ; utilisez des signatures numériques ; conservez une liste d'éditeurs approuvés. »

Il se peut que le niveau de sécurité par défaut soit trop fort pour exécuter les macros. Il convient alors de le baisser de la manière suivante :

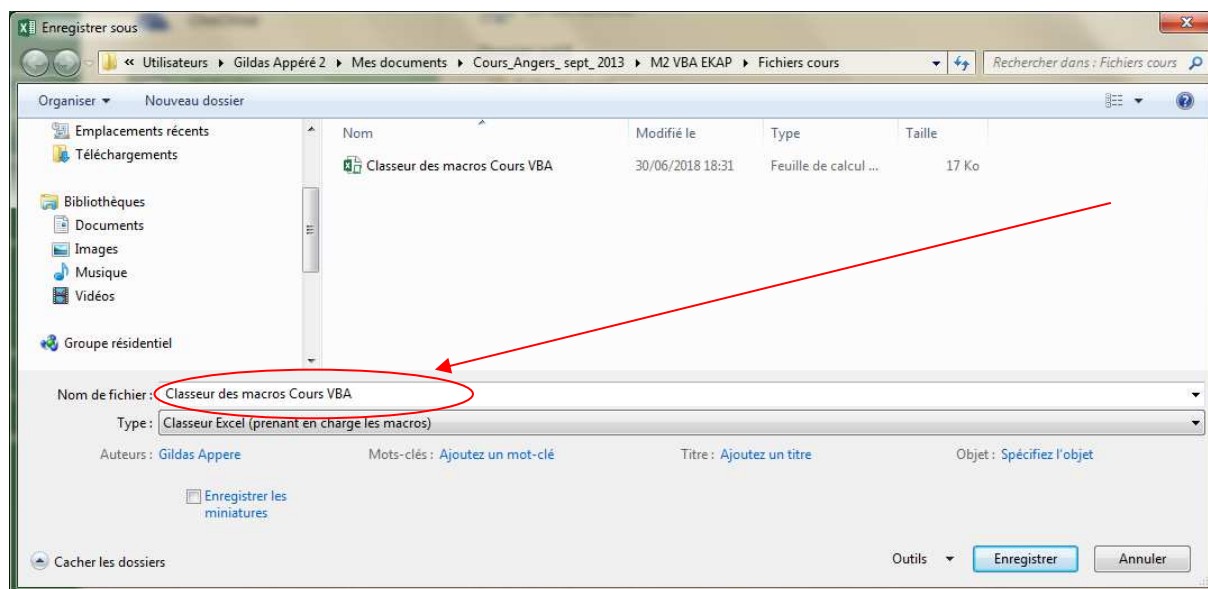
1. Dans l'onglet **Développeur**, cliquez sur **Sécurité des Macros**
2. Cliquez sur l'onglet **Sécurité**.
3. Cliquez sur le niveau de **sécurité** désiré



Lors de l'ouverture d'un classeur comprenant des macros, un message de sécurité devrait alors apparaître en fonction du niveau de sécurité choisi.

- **Sauvegarde des macros et des projets VBA (rattachés à un classeur)**

Attention : dès que vous travaillez sur des macros et / ou VBA, il faut enregistrer votre travail sous un classeur « **prenant en charge les macros** » sinon vous les perdez dès la fermeture du classeur ...



1.3. Enregistrement, exécution et accessibilité d'une macro simple

Pour comprendre et maîtriser les macros simples sous XL, on peut partir d'un exemple très simple consistant, en une seule commande, à :

- Mettre en gras
- Mettre en italique
- Centrer ...

... le contenu d'une cellule ou d'une plage de cellules sélectionnées.

❖ Etape préalable : création d'un document XL

- Ouvrir un classeur XL vierge ou le classeur « (1) *Classeur des Macros Cours VBA .xlsm* »
- Créer (ou utiliser) le tableau de calcul de coûts suivant (au moins les deux premières lignes)

Classeur des macros Cours VBA - Microsoft Excel

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1															
2		Unités produites	Coûts fixes	Coûts variables	Coût total	Coût marginal	Coût fixe moyen	Coût variable moyen	Coût total moyen						
3		0	200	0	200										
4		1	200	50	250	50	200,0	50,0	250,0						
5		2	200	75	275	25	100,0	37,5	137,5						
6		3	200	100	300	25	66,7	33,3	100,0						
7		4	200	140	340	40	50,0	35,0	85,0						
8		5	200	190	390	50	40,0	38,0	78,0						
9		6	200	250	450	60	33,3	41,7	75,0						
10		7	200	320	520	70	28,6	45,7	74,3						
11		8	200	400	600	80	25,0	50,0	75,0						
12		9	200	500	700	100	22,2	55,6	77,8						
13		10	200	650	850	150	20,0	65,0	85,0						

- **Enregistrer ce classeur** sur le bureau ou sur votre clé sous le nom « Calcul des Coûts unitaires »

❖ Enregistrement de la macro Gras Italique Centrer :

- **Avant de commencer l'enregistrement**, se placer (par exemple) sur la cellule B2 (Unités produites) afin de la rendre active
- Ouvrir une nouvelle macro : Développeur > Enregistrer une Macro

Calcul des coûts unitaires [Mode de compatibilité] - Microsoft Excel

	A	B	C	D	E	F	G	H	I	J	K	L	M
1													
2		Unités produites	Coûts fixes	Coûts variables	Coût total	Coût marginal	Coût fixe moyen	Coût variable moyen	Coût total moyen				
3		0	200	0	200								
4		1	200	50	250	50	200,0	50,0	250,0				
5		2	200	75	275	25	100,0	37,5	137,5				
6		3	200	100	300	25	66,7	33,3	100,0				
7		4	200	140	340	40	50,0	35,0	85,0				
8		5	200	190	390	50	40,0	38,0	78,0				
9		6	200	250	450	60	33,3	41,7	75,0				
10		7	200	320	520	70	28,6	45,7	74,3				
11		8	200	400	600	80	25,0	50,0	75,0				
12		9	200	500	700	100	22,2	55,6	77,8				
13		10	200	650	850	150	20,0	65,0	85,0				

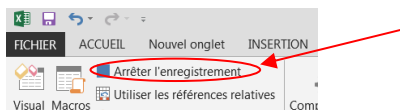
La Boîte de Dialogue suivante apparaît :

- Remplacer « Macro1 » par **un nom plus explicite** tel que : Gras_Italique_Centrer
- **Inform**er la case **Description** de manière relativement précise l'action réalisée par cette macro
- **Eventuellement : Créer une touche raccourci** Ctrl + b (attention : il faut être rigoureux concernant les raccourcis clavier car il n'est indiqué nulle part et peut écraser un raccourci déjà existant)
- **Enregistrer** cette macro dans **Ce classeur** (voir plus loin pour les explications)



- Cliquer OK

→ Dans la barre d'outil : l'apparition de **Arrêter l'enregistrement** signifie que l'enregistrement commence : **toutes les commandes** qui auront été exécutées jusqu'à l'arrêt seront intégrées dans cette macro ... donc ne pas oublier d'arrêter l'enregistrement dès que les actions désirées ont été enregistrées !



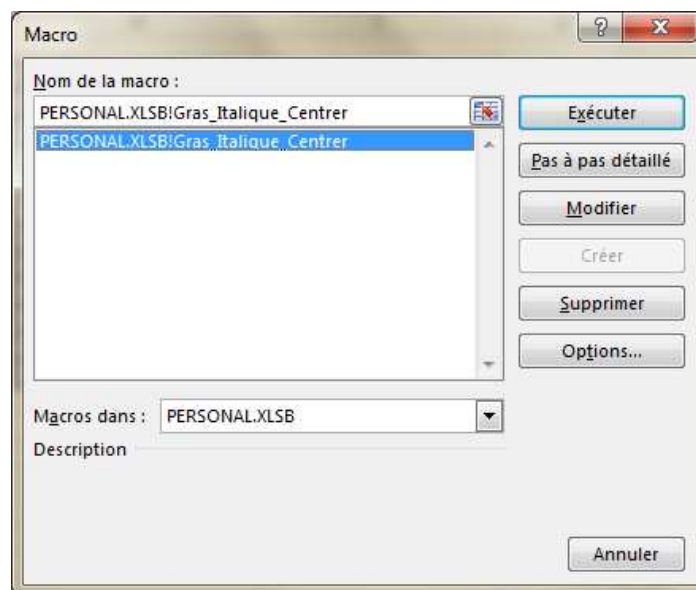
- Mettre en **gras, en italique et centrer** la cellule active B2 qui avait été sélectionnée juste avant le début de la macro
→ **Attention** : si, lors de l'enregistrement de la macro, vous n'étiez pas initialement sur la cellule B2 et vous cliquez alors sur B2, cette action « rendre B2 active » sera enregistrée, ce qui signifie que lorsque vous utiliserez cette macro sur n'importe quelle cellule (ex C4) , la macro va ordonner de ne mettre en forme que la cellule B2 et non la ou les cellules sélectionnées (ce qui n'est pas souhaitable!).
- Mettre **fin à l'enregistrement** en cliquant sur le carré bleu d'Arrêt

❖ Exécution de la macro Gras_Italique_Centrer

Pour vérifier que cette nouvelle macro fonctionne (et des 2 manières possibles), on peut l'appliquer à la plage de cellules (C2:I2) qui correspond aux intitulés des colonnes du tableau

1^{ère} méthode : via le Menu

- Sélectionner tout d'abord la cellule C2
- Exécuter la macro Gras_Italique_Centrer **via le menu** : Développeur > Macro> Macro dans : PERSONAL.XSLB
- Puis Exécuter la macro Gras_Italique_Centrer :



ça devrait fonctionner ...

2^{ème} méthode : Via le raccourci clavier

- Sélectionner ensuite la plage de cellules (D2 :I2)
- Exécuter la macro Gras_Italique_Centrer via **la touche de raccourci Ctrl + b**

ça devrait fonctionner ...

Remarque : quelle que soit la complexité de la macro, la procédure est toujours la même.

❖ Stockage et accessibilité d'une macro

L'accès ultérieur aux macros (par exemple si vous voulez les utiliser pour d'autres classeurs) aux macros est paradoxalement un peu plus complexe que la création des macros.

L'accessibilité d'une macro signifie : son exécution, sa modification ou sa suppression.

L'accessibilité d'une macro peut être :

- **globale** : on peut y accéder à partir de n'importe quel document
- **limitée** : on ne peut y accéder qu'à partir de documents spécifiques

Cette accessibilité est déterminée via **son stockage** qui est défini lors de la création de la macro (cf. exemple précédent).

Plusieurs possibilités sont envisageables :

a) Enregistrement de la macro dans le **classeur ouvert** sur lequel on travaille actuellement

Par exemple, si on travaille sur le classeur « *(1) Classeur des Macros Cours VBA .xlsm* », et qu'on y enregistre la macro « Rouge_Blanc » (mettant un fond rouge et les caractères en blanc) on aura :



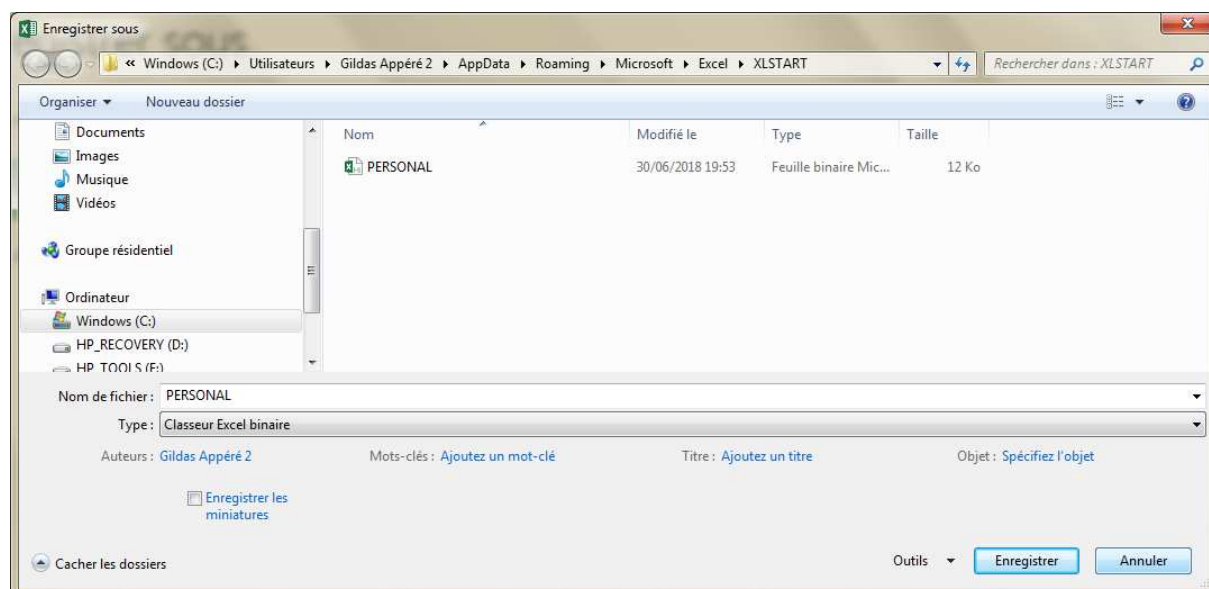
Conséquences :

- La macro enregistrée (ici Rouge_Blanc) n'est accessible que si le classeur « *(1) Classeur des Macros Cours VBA .xlsm* » est ouvert.
- La Macro Rouge_Blanc peut être exécutée sur un autre classeur, à condition que le classeur « *(1) Classeur des Macros Cours VBA .xlsm* » **soit ouvert**.

b) Enregistrement de la macro dans le classeur de macros personnelles

Si vous pensez que la macro que vous avez créée n'est pas spécifique à votre travail en cours (c.-à-d. attachée au classeur actuellement utilisé) mais **peut être utilisée pour d'autres travaux**, il n'est pas très pertinent de l'enregistrer sous le classeur en cours car, il vous faudra systématiquement ouvrir ce dernier (avec le risque de perdre cette macro si ce classeur est supprimé).

Dans ce cas, il faut enregistrer cette macro dans le **classeur de macros personnelles : PERSONAL.XLSB**, dont l'emplacement est quelque part sur le disque dur dans un sous-dossier (ex. C:\Users\Dupont\AppData\Roaming\Microsoft\Excel\XLSTART): la macro peut être alors utilisée et modifiée même à partir d'autres classeurs. Ce classeur est **généré la première fois** (sous le type *Classeur Excel Binaire*) que vous créez une macro dans le classeur de macros personnel.



L'intérêt de PERSONAL.XLSB est que ce classeur **s'ouvre ensuite automatiquement en mode masqué** (donc invisible pour l'utilisateur) lorsqu'on lance Excel, ce qui permet d'exécuter toutes les macros qui y sont enregistrées à partir de n'importe quel classeur sur lequel vous travaillez.

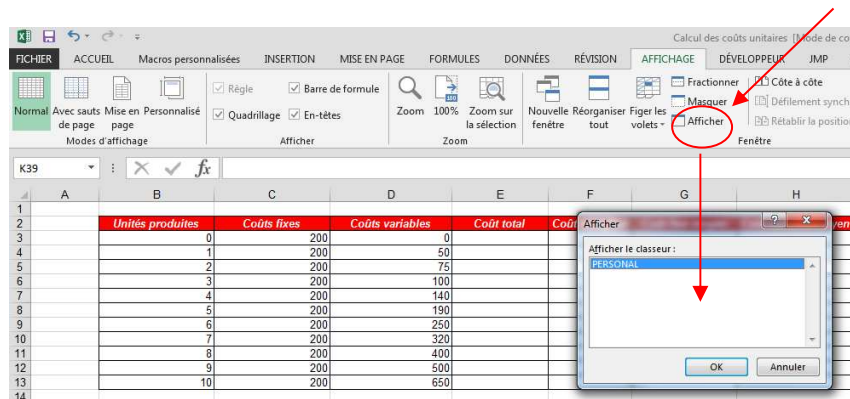
L'inconvénient : le classeur PERSONAL.XLSB s'ouvre automatiquement (en **mode masqué** ou visible) lors de l'utilisation d'EXCEL, avec toutes les macros qui y sont enregistrées, même si vous n'en avez pas besoin, ce qui risque éventuellement de ralentir inutilement le fonctionnement d'EXCEL.

Remarque :

Ce fichier PERSONAL n'est pas facilement accessible (c'est le but) afin d'éviter des manipulations accidentelles qui effaceraient ces macros personnelles qu'on peut supposer être importantes pour l'utilisateur.

Par la suite, afin de **modifier une macro personnelle**, il faut au préalable faire apparaître explicitement **PERSONAL.XLSB** :

→ Affichage > Afficher ... puis de cliquer OK concernant lorsque la fenêtre ci-dessous apparaît :



Attention : en affichant le classeur PERSONAL.XLSB vous serez tenté de le fermer une fois terminé.

- en faisant cela, le classeur PERSONAL.XLS n'est plus actif
- les **macros personnelles ne sont plus accessibles**
- Il faut donc rouvrir PERSONAL.XLSB (et le masquer éventuellement) : le plus facile c'est de fermer puis rouvrir XL et de rouvrir XL.
- Il vaut donc mieux **masquer** ce classeur (procédure inverse de la précédente consistant à faire afficher PERSONAL)

Attention : on peut aussi masquer n'importe quel classeur => en faisant cela, ce classeur malgré son ouverture n'est pas accessible, tant qu'on ne l'a pas rendu visible (attention donc aux fausses manipulations)

c) Enregistrement de la macro sur un classeur de macros complémentaires

Si vous êtes appelés à avoir un **recours intensif aux macros**, c'est la solution idéale afin d'éviter l'inconvénient associé au classeur PERSONAL.XLSB.

Exemple :

- on crée un nouveau classeur « (2) Calculs des variations.xlsm »

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	
1															
2		Unités produites	Coûts fixes	Coûts variables	Coût total	Coût marginal	Coût fixe moyen	Coût variable moyen	Coût total moyen						
3		0	200	0	200										
4		1	200	50	250	50	200,0	50,0	250,0						
5		2	200	75	275	25	100,0	37,5	137,5						
6		3	200	100	300	25	66,7	33,3	100,0						
7		4	200	140	340	40	50,0	35,0	85,0						
8		5	200	190	390	50	40,0	38,0	78,0						
9		6	200	250	450	60	33,3	41,7	75,0						
10		7	200	320	520	70	28,6	45,7	74,3						
11		8	200	400	600	80	25,0	50,0	75,0						
12		9	200	500	700	100	22,2	55,6	77,8						
13		10	200	650	850	150	20,0	65,0	85,0						
14															

- Enregistrer la macro ou les macros associées à ce classeur.

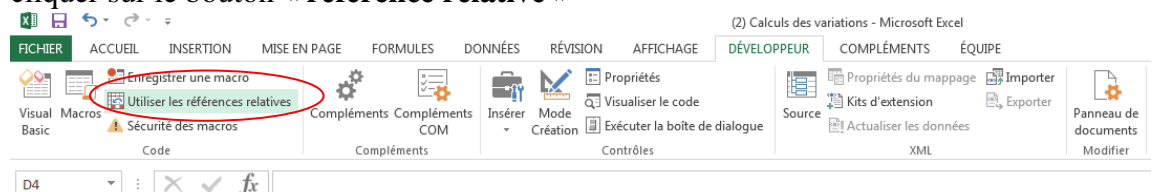
Par exemple, **on va créer 2 macros** : une macro calculant un taux de variation (nommée *taux_variation*) et une macro calculant le coefficient multiplicateur (nommée *coeff_multiplieur*) entre 2 données établies correspondant à deux dates (années) consécutives (et placées en colonnes)¹

→ Création de la macro *taux_variation*

:

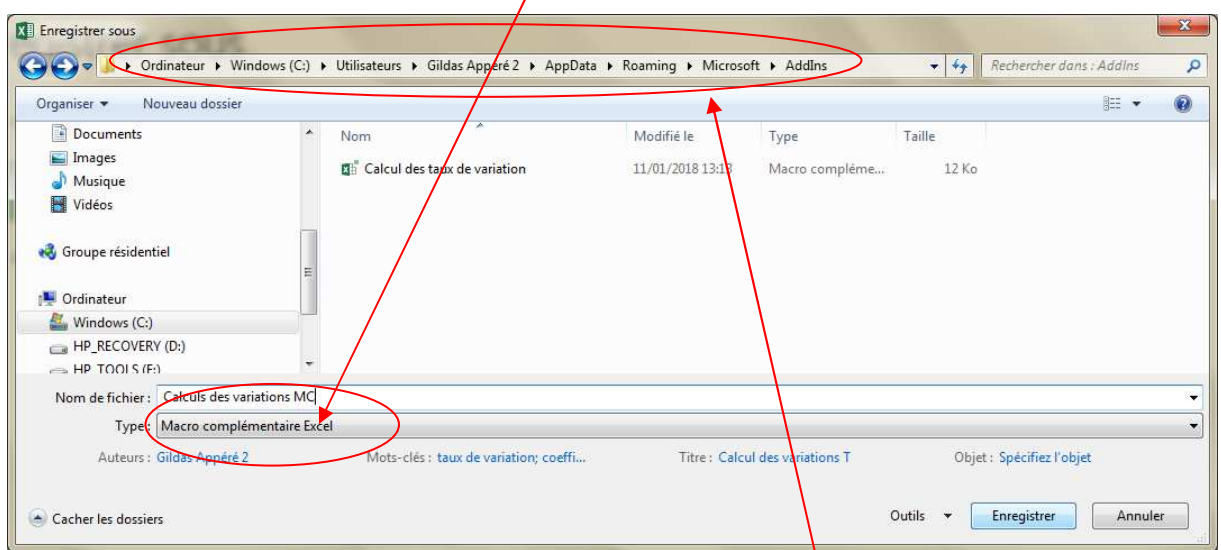
- Cliquer au préalable sur la **cellule D4** (pour la rendre active) puis commencer l'enregistrement dans le classeur actif de la macro dénommée « *taux_variation* »

- **Enregistrer la séquence** correspondant au calcul suivant :
- **Touche de raccourci = a**
- cliquer sur le bouton « **référence relative** »



¹ En fait, il serait plus pertinent de créer directement une **fonction** *taux_variation* ; ceci sera vu ultérieurement lorsqu'on abordera VBA.

- calculer dans la cellule D4 : $=(C4-C3)/C3$ (+entrée) et mettre la cellule en pourcentage (résultat affiché attendu : 3 %)
- **Vérifier** que la macro fonctionne en vous positionnant par exemple sur D6 (résultat affiché attendu : – 1 %)
- Faire la même chose avec la 2^{ème} macro *coeff_multiplyeur* (raccourci = b)
- **Enregistrer le classeur** « (2) *Calculs des variations* » sous le format habituel .xlsm (pour conserver une copie facilement accessible)
- **Enregistrer le classeur** « (2) *Calculs des variations Macro Comp* » sous le format de Macro complémentaires (**format .xlam**) qui est l'extension sous Excel qui désigne les classeurs comportant **les macros complémentaires**



Ce type de classeur est rangé par défaut dans un dossier spécifique de Microsoft complémentaires (ex.C:\Users\Dupont\AppData\Roaming\Microsoft\AddIns), peu accessible afin qu'il ne soit pas modifié accidentellement)

- **Difficulté** concernant la mise en œuvre effective des macros complémentaires :

La création d'une macro complémentaire ne suffit pas pour l'utiliser ensuite. Il faut au préalable l'**activer** ...

Exemple :

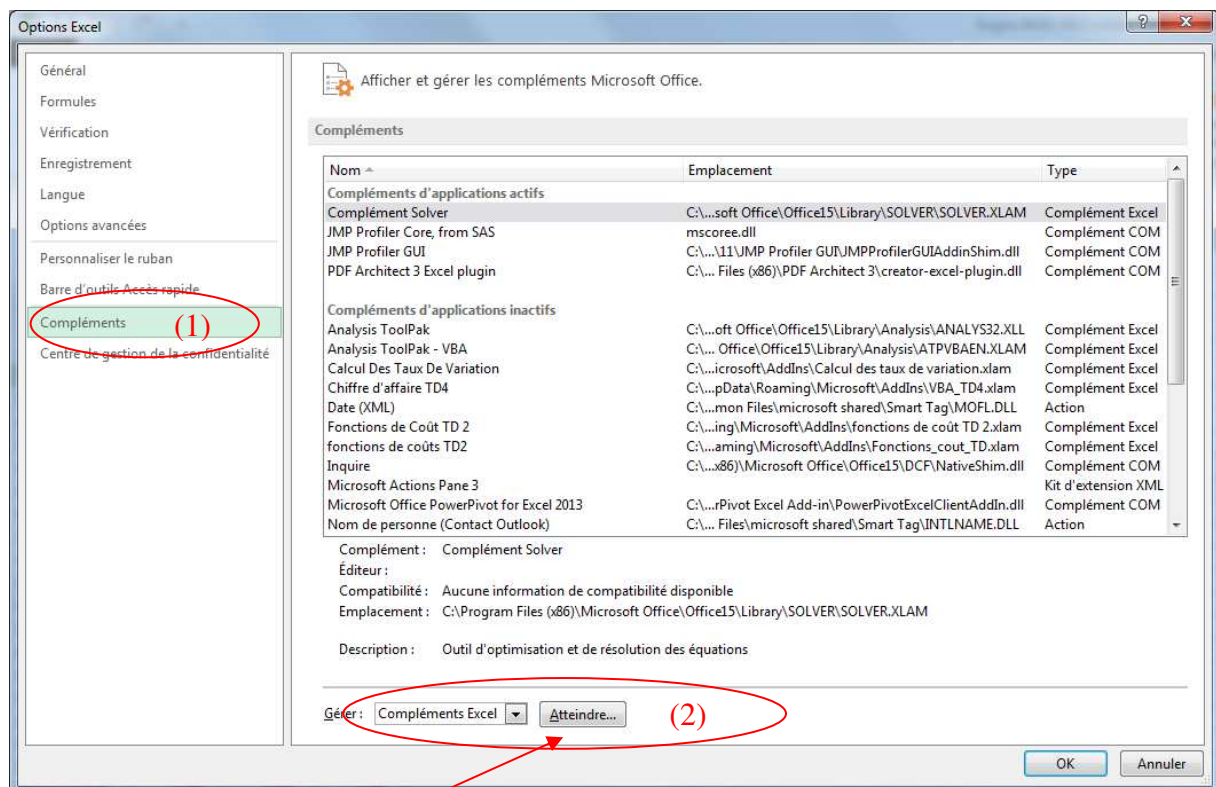
- **Fermer le classeur** « (2) *Calculs des taux de variation* » (.xlsm)
- **Ouvrir un nouveau classeur** et y inscrire deux nombres consécutivement en colonne, par exemple : 10 en B3 et 15 en B4
- Se situer en C4 et exécuter la macro *taux_variation* : rien ne devrait se produire car la macro complémentaire n'est pas activée (et le classeur source « (2) *Calculs des taux de variation* » est fermé).

=> Il faut donc activer au préalable cette nouvelle macro complémentaire ...

- Pour activer cette nouvelle macro complémentaire :

(Cette opération n'est à faire **qu'une fois** sauf si vous décidez par la suite de désactiver cette macro)

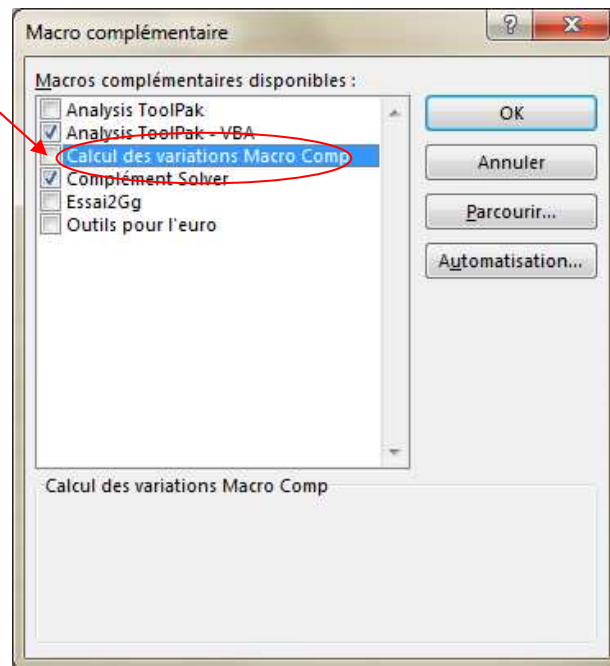
- Fichier > Options > Compléments > Compléments Excel > Atteindre



(3) Cliquer sur atteindre

- Dans la Boîte de Dialogue **Macro Complémentaire**, cochez la case *Calcul des variations Macro Comp*: toutes les macros associées à cet ensemble sont alors disponibles (on remarque au passage toutes les macros complémentaires disponibles telles que *Outils pour l'Euro*).

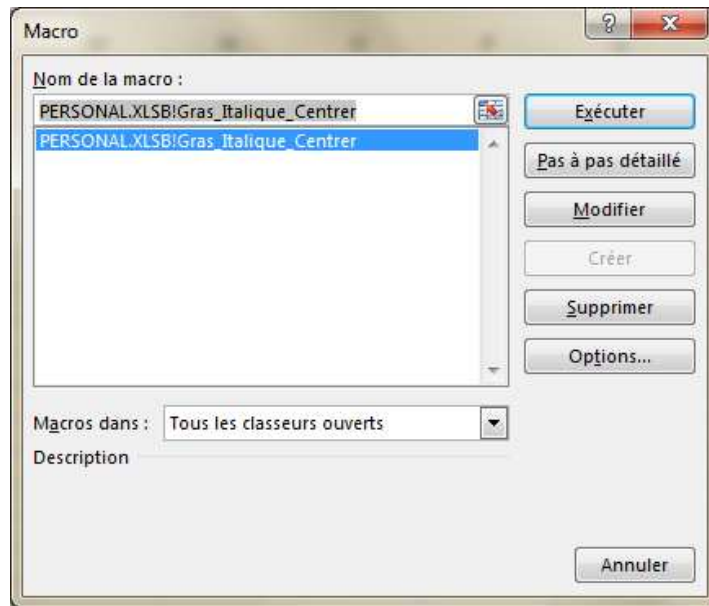
(4)



- **Vérifier** que les 2 macros taux_variation et coeff_multiplicateur sont maintenant **actives** en tapant de nouveau **Ctrl + a** et **Ctrl+b** sur un exemple simple
- Ces macro (ainsi que toutes les autres qui auraient été créées dans le classeur .xlam) sont actives dès que l'on ouvre Excel. Pour **les désactiver**, il suffit de faire l'opération inverse : Outil > Macros complémentaires ... puis décocher

Compléments :

Le problème des macros complémentaires, c'est **qu'elles ne semblent mobilisables qu'à partir d'un raccourci clavier** (d'après ce que j'ai pu voir). En effet, si on veut passer par le menu (Développeur > Macro >) on visualise :



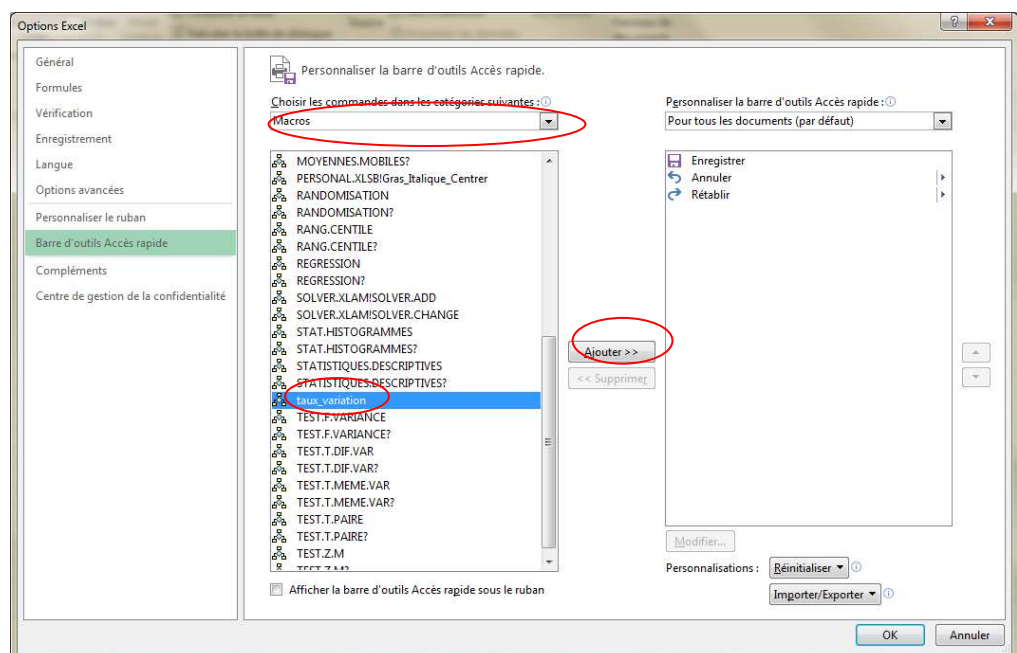
... **aucune trace** de la macro « *taux_variation* » et « *coeff_multiplicateur* » ! Ceci peut être assez gênant, car il faut alors connaître de mémoire le raccourci de ces macros (problématique lorsque le nombre de macros devient important).

Une solution possible est alors de créer, dès la création de la macro, **un bouton de commande** spécifique soit dans la **barre d'outils Accès rapide** (facile à faire) soit dans un nouveau groupe personnalisé (un peu plus long à faire)

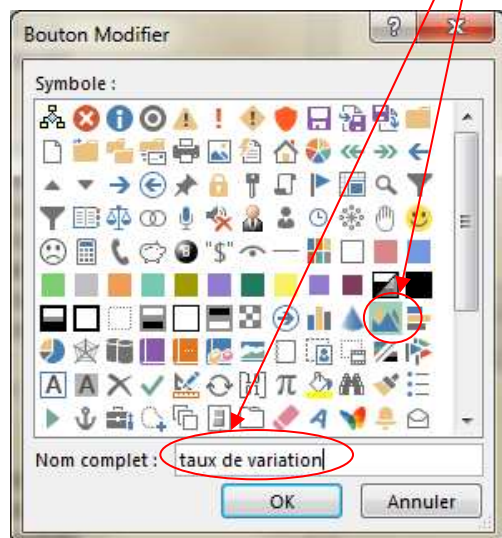
Procédure :

- Création du bouton **dans la barre d'outils Accès rapide**

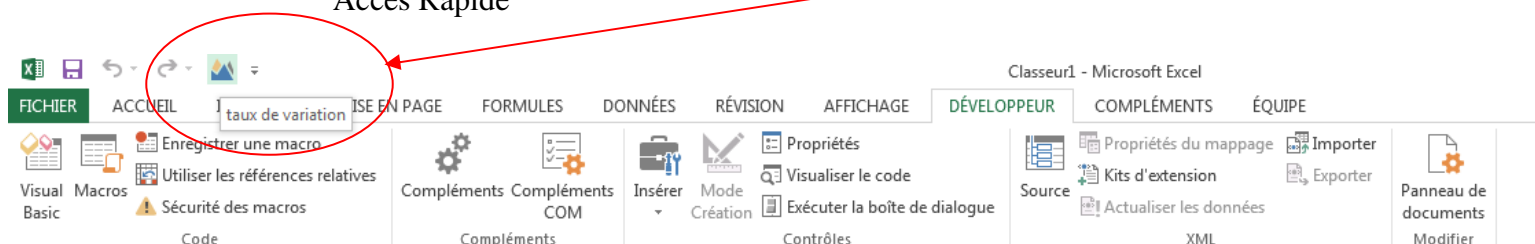
Fichier > Options > Barre d'outils Accès Rapide > Macro



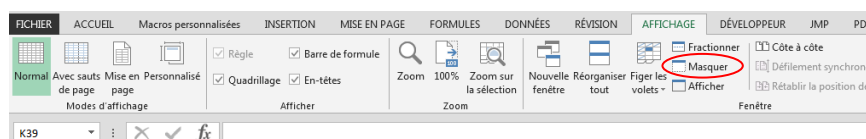
→ Ajouter *taux_variation* puis modifier éventuellement son nom et son icône à l'aide du mouton **Modifier** (faire ensuite la même chose pour les autres macros telles que *coeff_multiplieur*)



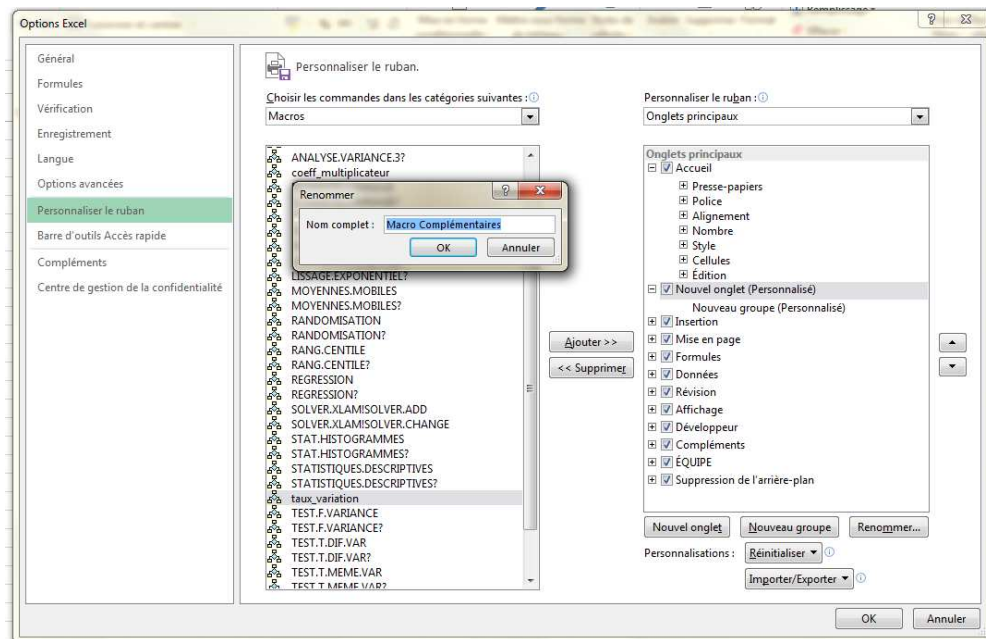
La Macro *taux_variation* est maintenant disponible dans la barre d'outils Accès Rapide



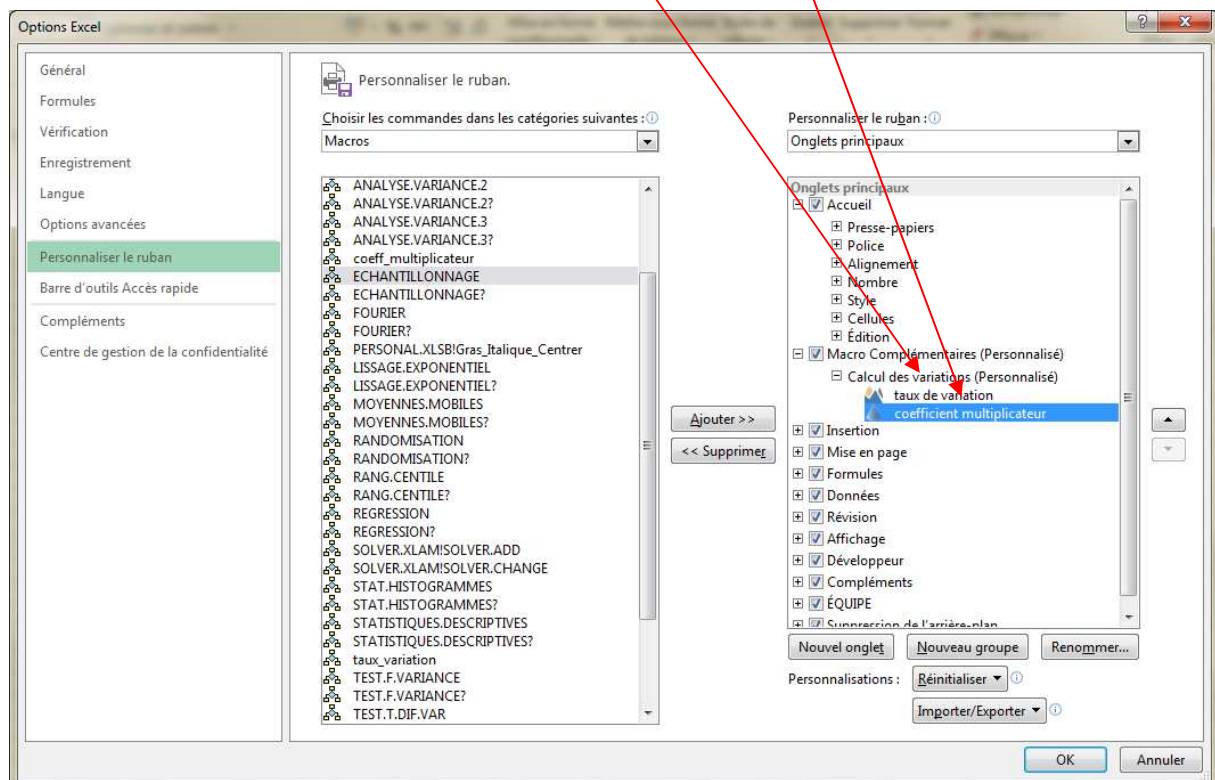
Remarque : en testant ce bouton sur un nouveau classeur, il se peut que cela fasse ouvrir et apparaître le classeur *calcul des variations Macro Comp.xlam* → dans ce cas, il convient de le masquer (une fois pour toute) et d'enregistrer cette modification.



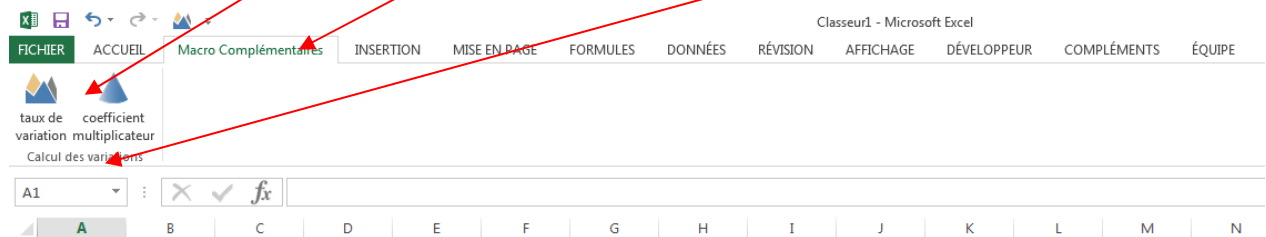
- Création du bouton **dans un groupe personnalisé et dans un nouvel onglet**
 - Fichier > Options > Personnaliser le ruban
 - problème : **on ne peut pas directement insérer cette macro** dans un onglet
 - on crée dans au préalable un **groupe personnalisé** éventuellement dans un **nouvel onglet** :
Exemple : on peut créer l'onglet **Macro Complémentaire** : Nouvel Onglet > ...



- on crée un **groupe personnalisé** associé aux formules relatives aux calcul des variations : Nouveau groupe > ... (on peut changer l'icône et le nom)
- on ajoute **les macro personnalisées** *taux_variation* et *coeff_multiplyeur* dans ce nouveau groupe (en modifiant si on le souhaite leur nom affiché et leur icones)



On obtient alors le **nouvel onglet** *Macro Complémentaires* qui inclut le groupe *Calcul des variations* regroupant nos 2 macro



- Vérifier que ces boutons fonctionnent en refermant puis rouvrant XL en faisant un calcul simple de 2 nombre situés en colonne

1.4. Références absolues et références relatives lors des enregistrements de macros

Lorsqu'on enregistre puis exécute une macro, la question est de savoir quelle(s) sont les cellules concernées par cette macro.

Par défaut, Excel mémorise les **références absolues**, ce qui signifie que :

Si au cours de votre enregistrement on clique sur la cellule B2, puis la cellule D3, Excel mémorise effectivement les cellules B2 et C2 (**références absolues**) et non pas simplement deux cellules telles que la deuxième est située une case à droite et une case au dessous de la première (**références relatives**).

Or, il est souvent utile de recourir aux références **relatives**.

Solution : utiliser les **références relatives au début de l'enregistrement d'une macro portant sur des manipulations faites sur des cellules**

1.5. Ecriture et modification d'une macro : une introduction à la programmation sous VBA

Pour modifier le fonctionnement d'une macro, il faut en modifier **la syntaxe écrite en VBA ... encore faut-il pouvoir y accéder !**

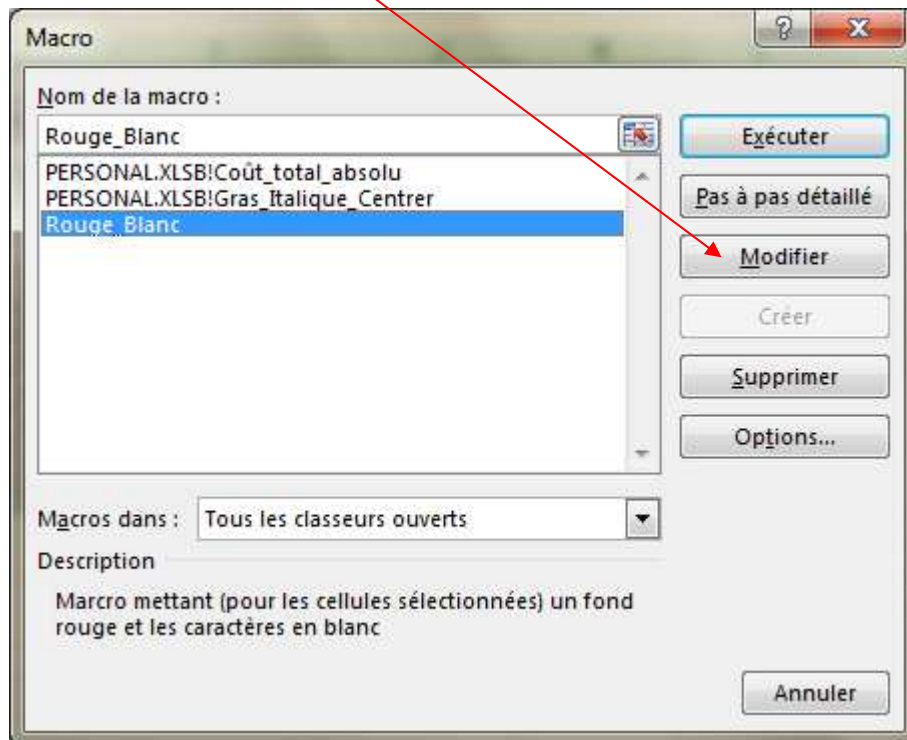
1.5.1. Comment accéder à une macro déjà enregistrée ?

L'accès à la fenêtre de Codes d'une macro dépend de la manière dont cette macro a été enregistrée :

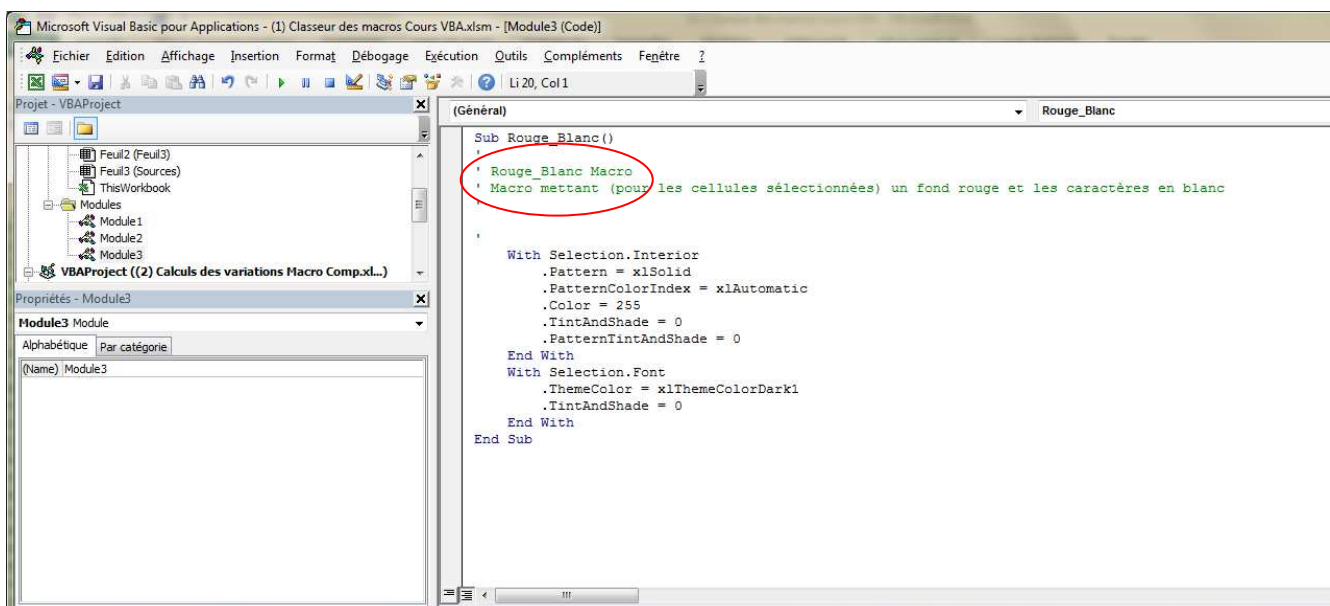
a) Si la macro a été enregistrée dans un classeur particulier

Exemple : on veut accéder à syntaxe de la macro *Rouge_Blanc* associée **uniquement** au classeur (1) *Classeur des Macro Cours VBA.xlsm*

- Ouvrir le classeur (1) *Classeur des Macro Cours VBA.xlsm*
- **Sélectionner** la macro *Rouge_Blanc* en passant par le menu
- Cliquer sur le bouton **Modifier**



La **syntaxe VBA** de la macro *Blanc_Rouge* devient visible dans la fenêtre VB suivante (Editeur VB, VBE)

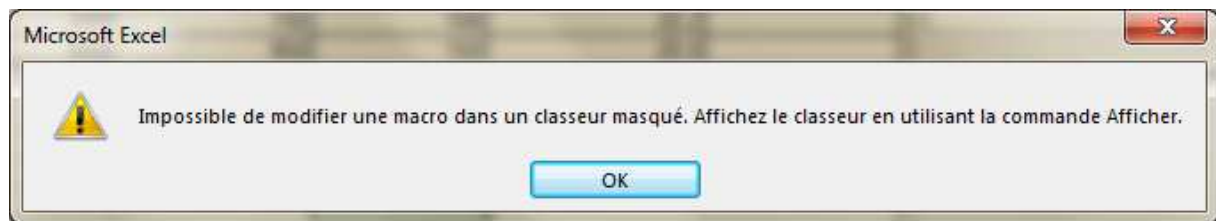


b) Si la macro a été enregistrée dans le classeur PERSONAL.XLSB

Exemple : on veut accéder à syntaxe de la macro *Gras_Italique_Centrer* associée au classeur de macros personnelles PERSONAL.XLSB

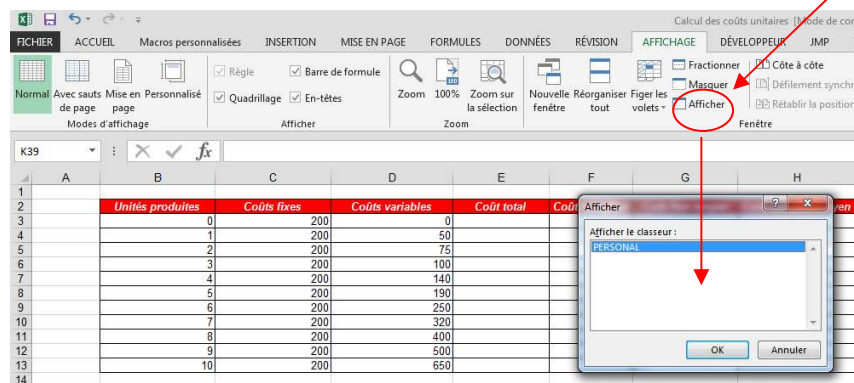
- Ouvrir le classeur (1) *Classeur des macros Cours VBA.xlsm*
- **Sélectionner** la macro *Gras_Italique_Centrer* en passant par le menu
- Cliquer sur le bouton **Modifier**

Il est fort probable que la boîte de dialogue suivante s'affiche :



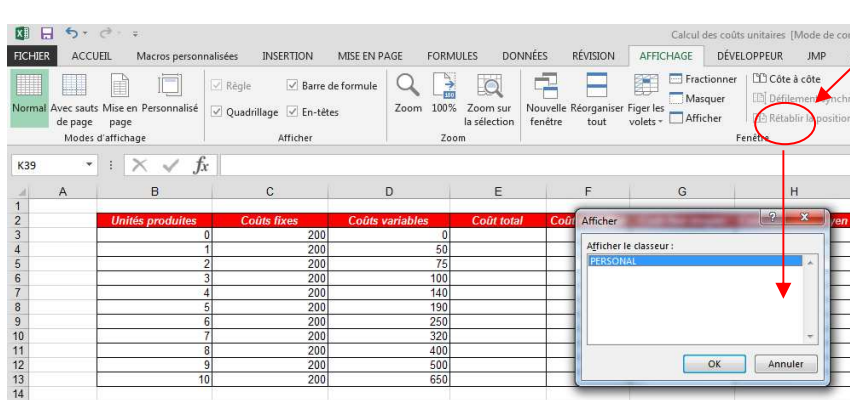
En effet, pour modifier cette macro, **il faut que le classeur PERSONAL.XLSB soit apparent.**

Remarque : Par la suite, pour faire apparaître **PERSONAL.XLSB** afin de modifier une macro enregistrée dedans (voir plus loin), il suffit de faire : Affichage > Afficher ... puis de cliquer OK concernant lorsque la fenêtre ci-dessous apparaît :



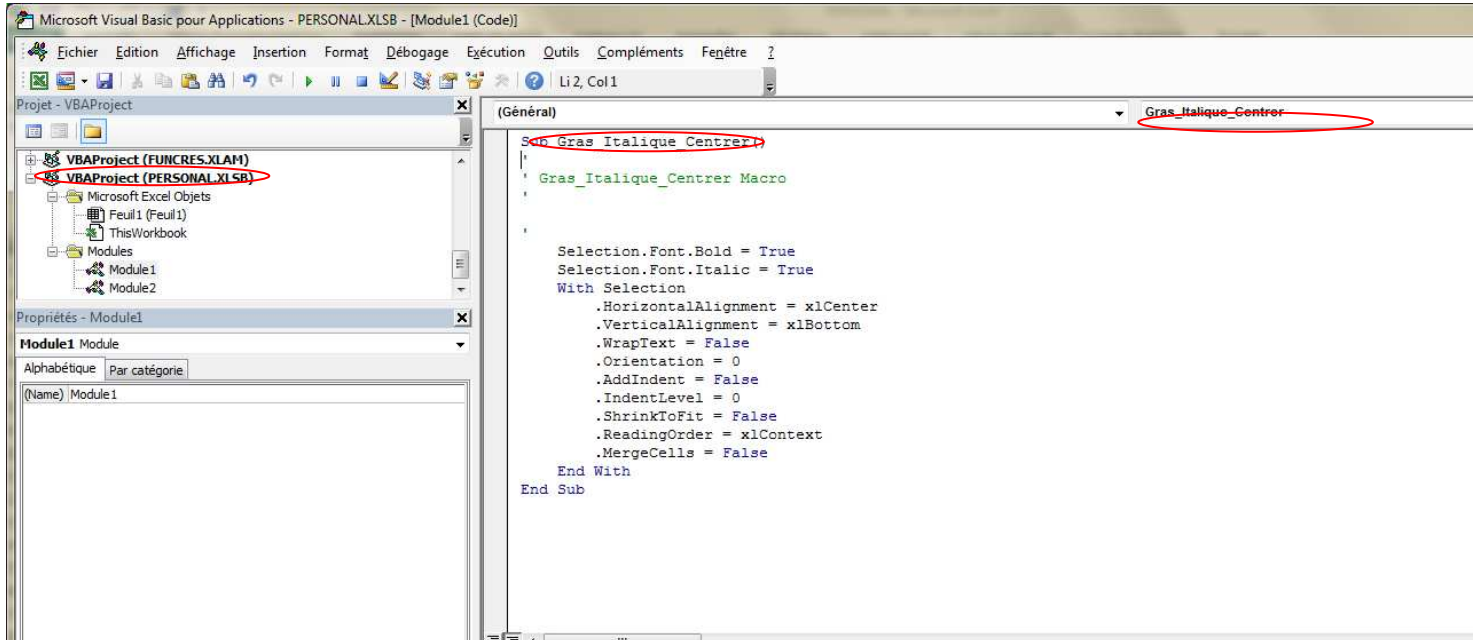
Procédure :

- faire apparaître **PERSONAL.XLSB** : Affichage > Afficher ... puis de cliquer OK concernant lorsque la fenêtre ci-dessous apparaît :



(Rappel : par la suite, surtout ne pas fermer ce classeur PERSONAL → il faut le masquer, pas le fermer)

- revenir sur le classeur (1) *Classeur des macros Cours VBA.xlsm*
- Accéder à la syntaxe de la macro *Gras_Italique_Centrer* via le menu



1.5.2. Comment modifier une macro déjà enregistrée : approche intuitive

Une approche intuitive (*bidouille* ?) pour comprendre comment fonctionne une macro et plus généralement comment fonctionne la syntaxe VBA est de créer une macro simple avec

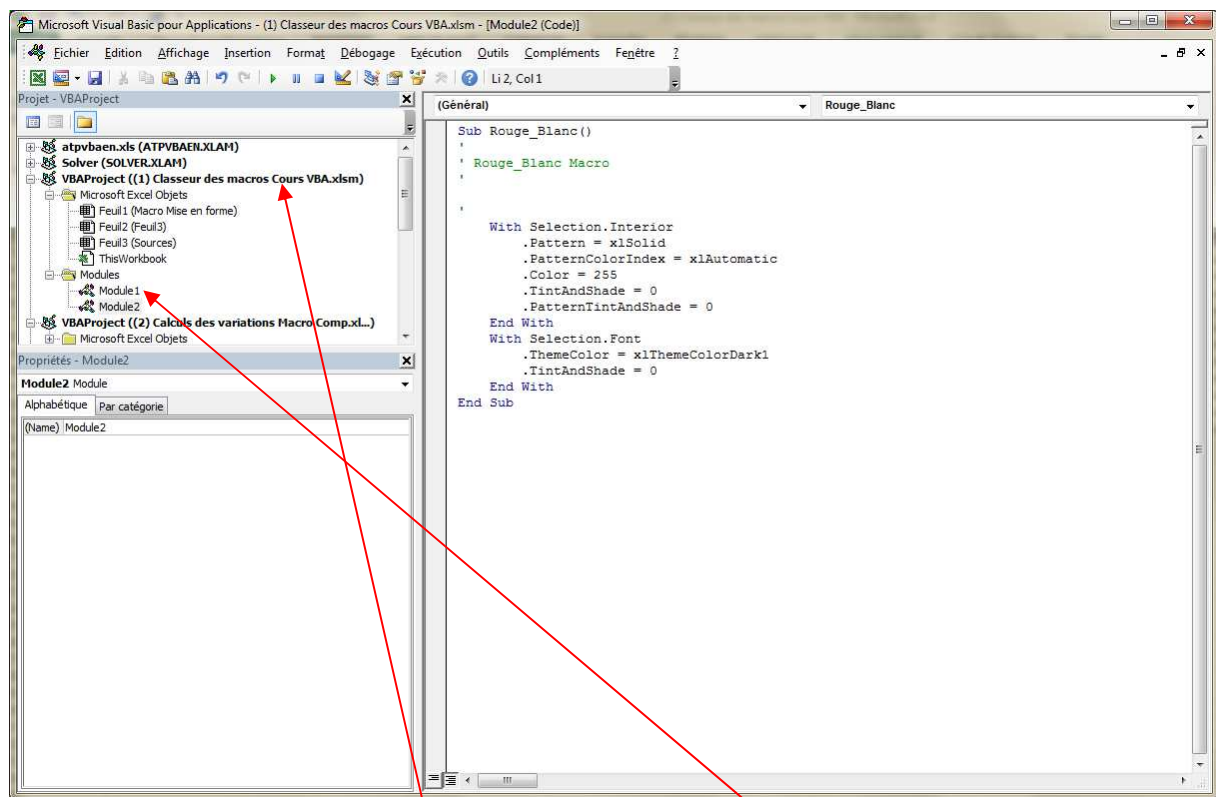
différentes commandes clairement spécifiées (ex. mettre en gras, centrer, ...) puis d'entrer dans la syntaxe de cette dernière et :

- d'observer comment une commande est traduite en code VBA
- faire quelques modifications de code et de « voir ce qui se passe » sur le fonctionnement de la macro

Exemple :

- Ouvrir le classeur (1) *Classeur des macros Cours VBA.xlsm*
- Accéder aux codes de la macro *Rouge_Blanc*

L'éditeur VBE indique la syntaxe suivante :



On observe que cette macro est enregistrée dans le **Module1** du **projet VBA** associé au classeur (1) *Classeur des macros Cours VBA.xlsm*

Les **éléments en vert** (précédés d'un accent en début de ligne) sont des **commentaires et informations** pour l'utilisateur telles que :

- Le nom de la macro
- Le commentaire inscrit lors de la création de la macro
- Le raccourci clavier associé à cette macro

Les éléments en bleu et noir sont les codes VBA dont la terminologie anglophone est souvent relativement explicite. Par exemple :

- **Selection.Interior** \Rightarrow concerne (pour les lignes en dessous indentées) le remplissage de la plage de cellules sélectionnée
- **.Color = 255** \Rightarrow concerne la couleur de ce remplissage ;
 - on sait donc que 255 correspond à la couleur rouge vif
 - on peut donc « expérimenter » en remplaçant 255 par 800 et faire exécuter la macro pour observer le changement de couleur (codes couleur : RVB = un peu compliqué Rouge vif \Rightarrow R= 255 V= 0 B = 0)
- idem pour la couleur de la police : **Selection.Font**.

Pour poursuivre dans cette approche intuitive, on peut faire la procédure inverse : afin d'observer comment traduire en codes VBA une série de commandes désirées :

- on crée la macro permettant d'exécuter les commandes désirées
- on observe ensuite les codes VBA correspondants qui viennent d'être créés.

Exemple : on désire connaître les codes VBA associés au changement de Police (ex. mettre en Police Verdana).

- On crée (de manière très provisoire car complètement inutile) la macro mettant en Police Verdana.
- On observe les codes correspondant, il s'agit de : `Selection.Font.Name = "Verdana"`

Grâce à cette procédure, on peut rapidement accéder à la syntaxe VBA désirée (pour des commandes simple, en tout cas).

2. La programmation sous VBA

La maîtrise des macros est un outil puissant pour automatiser des tâches sous Excel (mais aussi Word, Access, ...) et pour améliorer la productivité.

Cependant, créer des macros ne constitue pas véritablement une démarche de programmation, même si en « éditant » les codes de ces macros, on a pu avoir un aperçu intéressant du langage VBA et de son mode de fonctionnement.

En effet, pour qu'il y ait programmation sous VBA, il faut qu'il y ait une véritable démarche d'algorithme avec, notamment, le recours fréquent à des boucles répétitives des tests etc.

2.1. Qu'est-ce que VBA ?

Définition :

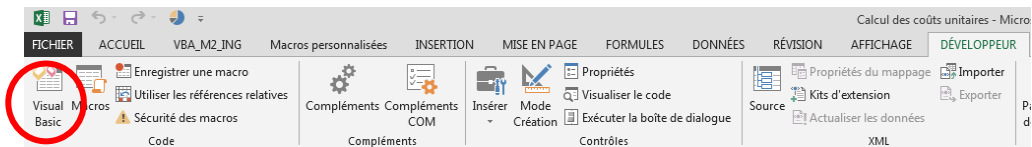
- VBA = *Visual Basic for Applications*
- C'est un environnement de **développement** qui a été, à partir de 1997, intégré dans la suite bureautique de Microsoft (Excel, Word, ...) ⇒ dans le cas présent, **l'application hôte est Excel**
- Son but : adapter via la programmation, de manière très fine, le fonctionnement de Excel aux besoins spécifiques de l'utilisateur, au-delà des possibilités (limitées) offertes par les macros ⇒ il s'agit donc véritablement d'adapter le fonctionnement d'Excel à ses propres besoins, lorsque ces derniers deviennent très spécifiques.
- VBA est un **langage de programmation orienté objet (POO)** : en clair, VBA **manipule des objets** (ayant des attributs spécifiques) tels que des cellules, des feuilles de calculs, des graphiques ...

2.2. L'accès à VBA : l'éditeur VBE (*Visual Basic Editor*)

VBE est l'application qui permet d'accéder aux codes des macros (cf. supra) et à la programmation VBA.

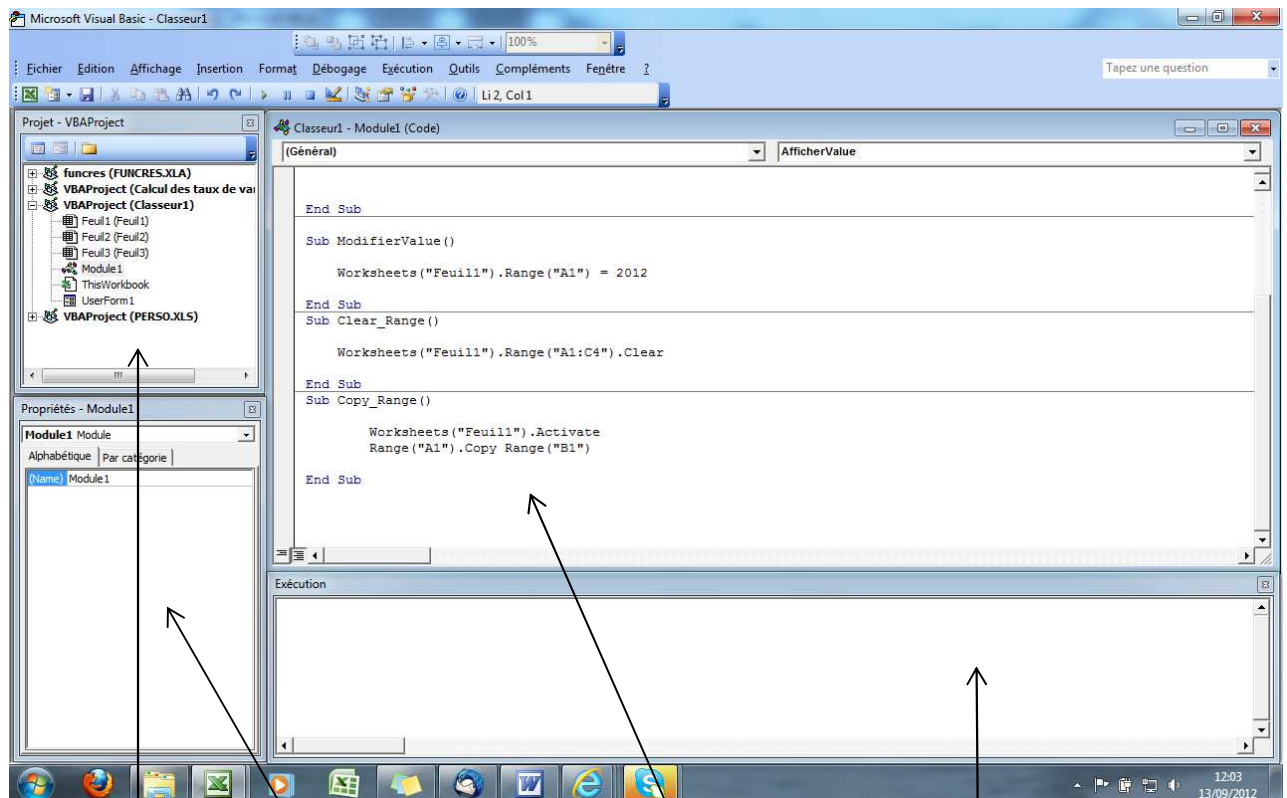
VBE ne peut pas être ouvert indépendamment de son application, à savoir ici Excel. Il existe divers moyens d'y accéder :

- via une macro qu'on va modifier
- via le bouton Visual Basic situé dans l'onglet **Développeur**



- via Alt + F11

La fenêtre suivante apparaît (à quelques choses près) :



Explorateur
de projet

Fenêtre de
propriétés

Fenêtre de
Codes

Fenêtre
d'exécution

Les **fenêtres non apparentes** peuvent être affichées via le menu **Affichage**

2.2.1. L'explorateur de projets :

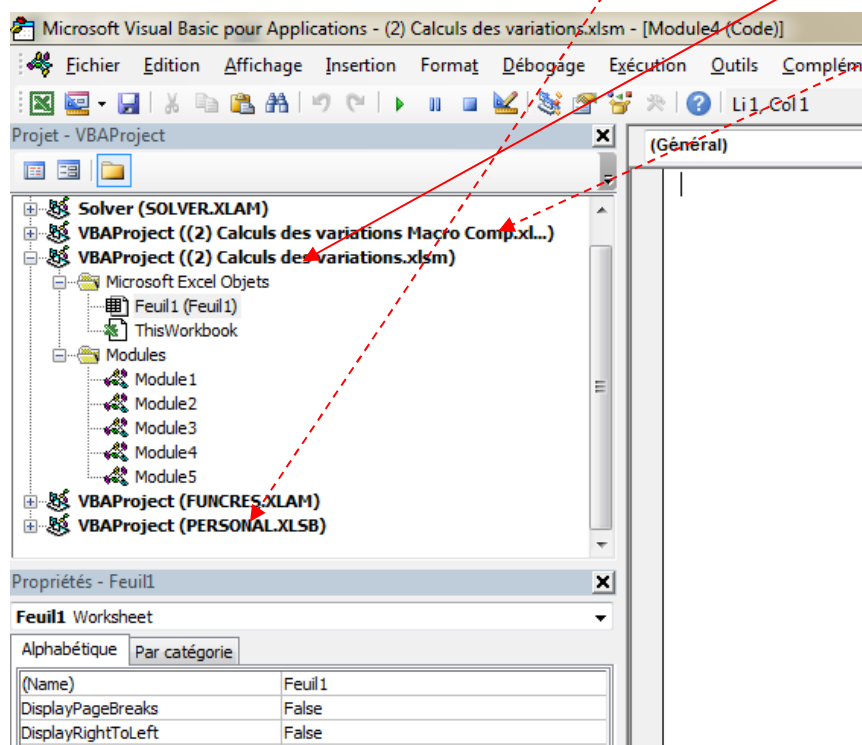
L'explorateur de projet permet de visualiser **les différents projets VBA ouverts** ainsi que les différents éléments qui les composent.

Un **projet VBA (VBAproject)** est un **ensemble d'objets** c'est-à-dire, en fait, un classeur composé lui-même de différents objets, en particulier :

- de **feuilles**
- de **modules** éventuels (si on en a créés, par exemple via une macro)
- de **formulaire (Userform)** éventuels (si on en a créés)

En double-cliquant sur un de ces éléments, on fait alors apparaître sa **fenêtre de propriétés** ainsi que sa **fenêtre (de modules de) codes** correspondantes.

Attention : la fenêtre (de module) de codes affichée par défaut, n'est pas forcément celle qui correspond à l'objet sélectionné. Pour être certain d'afficher le bon module, il vaut mieux double cliquer sur l'objet sélectionné dans la fenêtre de l'explorateur de projets. Par exemple, si je désire travailler sur le classeur (2) *Calcul des variations.xlsm*, je clique dessus pour être certain d'être sur celui-là et non sur un des autres classeurs ouverts (en particulier ceux qui s'ouvrent automatiquement, tels que PERSONAL.XLSB ou un des classeurs de Macro complémentaires)



2.2.2. Les Modules

- Définition : Les **modules** sont les lieux où sont **stockés** un ensemble de **procédures** (macros, fonctions) d'un projet (classeur Excel).
- Création, exportation, importation, suppression d'un module :
 - **Création automatique :** Un module est créé **automatiquement** (sans forcément utiliser le VBE) dès qu'on a créé une macro dans le classeur.
 - **Création « volontaire » :** Insertion > Module
 - Faire attention à la sélection préalable du bon projet VBA i.e. choisir le classeur désiré.

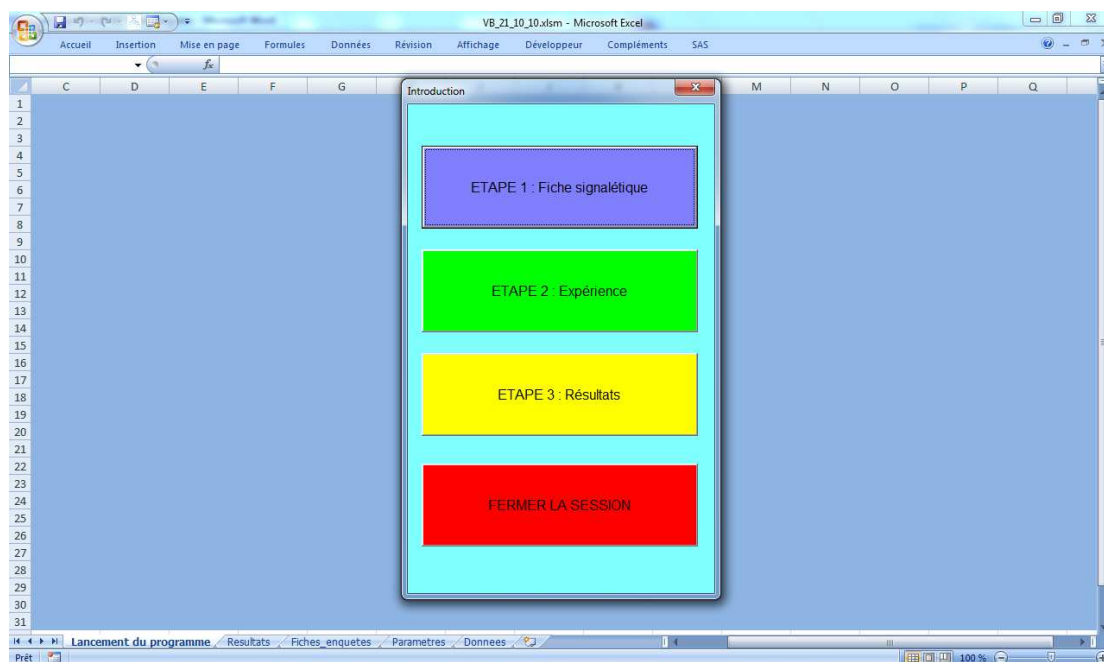
➤ **Exportation et importation d'un module :**

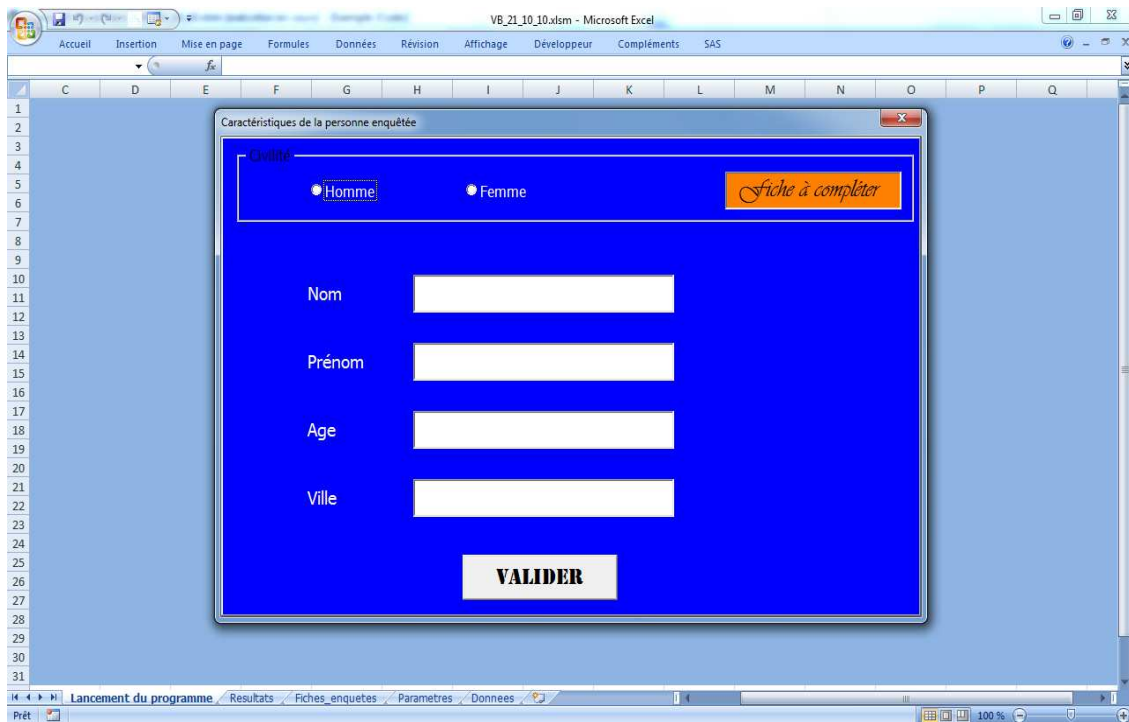
- **Exportation :** il s'agit d'enregistrer un module afin de l'utiliser dans un autre projet. Il suffit de cliquer-droit sur le module, cliquer sur exporter, puis de suivre les instructions. Ce module sera enregistré en format *.bas*
- **Importation :** il s'agit d'utiliser, dans un projet VBA, un module ayant été enregistré au cours d'un autre projet. Il suffit de cliquer droit sur le projet de destination, de cliquer sur importer, puis de suivre les instructions.

➤ **Suppression d'un module :** Cliquer-droit sur le module à supprimer puis supprimer) ⇒ VBA vous demandera toujours d'exporter ce dernier pour en faire une sauvegarde.

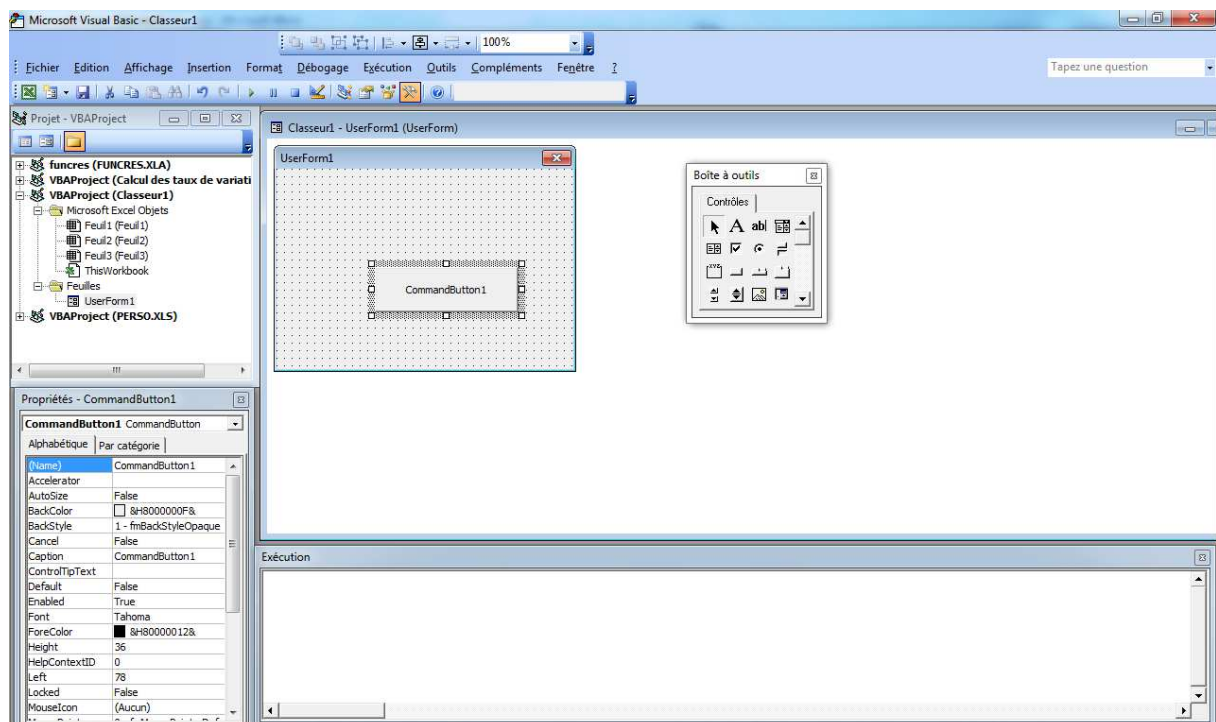
2.2.3. Les Userforms ou formulaires ou boîtes de dialogues personnalisées

- Ce sont des boîtes de dialogue personnalisées, **indépendantes des feuilles XL** (i.e. elles ne sont pas ancrées dans une feuille).
- Elles permettent une **interface** entre l'utilisateur et Excel pour recueillir différentes informations qui pourront être utilisées lors d'un programme VBA ou guider l'utilisateur.





- Leur apparition et leur fermeture sont commandées via des procédures écrites en VBA
- Création d'une Userform : Insertion > Userform



On peut alors modifier cette **Userform** (Couleur, taille, etc.) et y ajouter des éléments tels que des boutons auxquels sont rattachés des **procédures** codées en VBA (Sub ... End) provoquant les événements voulus par le concepteur du programme.

2.2.4. Fenêtre de codes (ou fenêtre de modules) :

Cette fenêtre contient les codes VBA des procédures (macros, fonctions) associées à tout élément du projet (module, Userform, Feuille) sélectionné : ces codes sont visibles en double cliquant sur le module désiré.

2.3. La programmation orientée objet (POO) : quelques éléments

Préambule : *Il ne s'agit pas ici d'un cours sur la POO : vous pourrez par vous-même, si cela vous intéresse, faire des recherches sur cet aspect.*

2.3.1. La notion d'objet en **POO** : une analogie avec la voiture²

La voiture en tant qu'**objet** désigne une chose ayant des **propriétés** bien précises (carrosserie, 4 roues, 1 moteur, ...) et des **méthodes** bien précises (démarrer, freiner, accélérer, tourner à droite, ...). Quand on parle de l'**objet Voiture**, il ne s'agit donc pas de la Citroën C3 rouge de M. Dupont en particulier, mais bien de la **Voiture** (au singulier) en général en tant qu'objet distinct de l'**objet Camion** (avec des similitudes concernant ses propriétés et ses méthodes) ou de l'objet télévision (aucunes similitudes concernant les propriétés et méthodes).

- On parlera également de la **classe Voiture** pour désigner de manière formelle ce qui définit précisément une voiture.
- A l'inverse, pour désigner les différentes voitures existantes (les voitures particulières de M. Dupont, Mme Durand etc.) on parlera de **Collection de Voitures**.

2.3.2. Objets, Collections d'objets, hiérarchie sous VBA pour Excel

2.3.2.1. Les objets (ou classes d'objets) dans VBA pour Excel

Excel propose plus de 100 **objets** tels que :

- la plage de cellules (**Range**)
- la feuille de calcul (**Worksheet**)
- le classeurs (**Workbook**)
- la fenêtres (**Window**)
- le graphique (**Chart**)
- Excel lui-même (**Application**)
- ...

Un objet est un donc **terme générique** pour désigner un **élément** ayant des propriétés et des méthodes propres à cet objet. Par exemple, l'objet **Feuille** désigne de manière générique un élément de l'**Application Excel** ayant des **propriétés** spécifiques (ex. la feuille contient des plages de cellules (qui sont elles-mêmes des objets) et des **méthodes** spécifiques (ex. **Ajouter** une feuille (Add)).

Une grande partie de ces objets sont **manipulables** avec des codes VBA : on peut ainsi écrire des lignes de code destinées respectivement à :

² Exemple inspiré du manuel *Excel et VBA*, Mickael Bidault, Pearson

- **Ouvrir** un classeur
- **Sélectionner** une feuille, ou une plage de cellules, ...
- **modifier** le contenu d'une cellule etc.

2.3.2.2. Collection d'objets

Une **collection d'objets** correspond à l'ensemble des objets (au pluriel) ayant les caractéristiques (propriétés et méthodes) définies par la classe de l'objet auquel ils appartiennent.

Exemples de collections d'objets :

- **Workbooks** : collection de tous les objets **Workbook** (classeur) actuellement ouverts
- **Worksheets** : collection de tous les objets **Worksheet** (feuille) actuellement ouverts dans un workbook particulier.
- **Charts** : collection de tous les objets **Chart** (graphique) contenus dans un classeur donné

On peut **dénombrer le nombre d'objets** présents dans la **collection** considérée par la commande `.Count` précédée du type d'objets (au pluriel) à dénombrer (ex. **Worksheets.Count**).

2.3.2.3. Faire référence à des objets

Pour **manipuler** un objet particulier ou toute une collection d'objets, il est nécessaire de pouvoir clairement les identifier (les référencer) précisément.

Par exemple, concernant les feuilles de calcul, si on veut travailler :

- sur la **collection entière** d'un objet : on fera référence à **Worksheets**
- sur une **feuille particulière** (ex. Feuil1) de la collection : on fera référence à **Worksheets("Feuil1")** ou **Worksheets(1)** si la feuille 1 est la première des objets en partant par la gauche

2.3.2.4. Hiérarchie des objets et navigation dans la hiérarchie

Un objet peut contenir différents objets qui contiennent eux-mêmes d'autres objets etc. : il s'agit d'une **hiérarchie d'objets** emboîtés les uns dans les autres.

Ainsi l'objet **Range** (plage de cellule) est inclus dans l'objet **Worksheet** (feuille de calcul), inclus dans l'objet **Workbook** (Classeur), lui-même inclus dans l'objet **Application** (qui est Excel dans le cas présent).

On dit alors que l'objet **Workbook** est le **conteneur** de l'objet **Worksheet**, qui lui-même le **conteneur** de l'objet **Range**.

Pour **naviguer dans la hiérarchie**, il faut naviguer d'amont en aval lorsqu'on écrit une ligne de codes (de gauche à droite) :

Exemples :

- Référence du classeur *Budget.xlsm* :

Application.Workbooks('Budget.xlsm')

Ce qui signifie que *Budget.xls* est un élément de la collection **Workbooks** (collection des classeurs ouverts) : `.Workbooks('Budget.xlsm')`, collection qui est elle-même incluse dans l'Application (Excel).

- Référence de la cellule A1 de la feuille *Feuille 1*, du classeur *Budget.xlsm* :

Application.Workbooks('Budget.xlsm').Worksheets('Feuille 1').Range('A1')

Remarques :

- Une cellule seule est codée comme une plage de données **Range** à une seule cellule : par exemple `Range('A1')` ; en fait il existe aussi un code spécifique **Cells** (voir plus loin).
- Un objet (au singulier), en tant que tel, n'apparaît jamais dans les lignes de code. C'est la collection d'objets correspondante (au pluriel) qui est toujours écrite.
- La dénomination **Application** est le plus souvent inutile car le programme est enregistré dans Excel et donc attribué par défaut Excel comme application. Cette dénomination est nécessaire lorsqu'on élabore un programme VBA utilisant plusieurs applications différentes (Word, Excel etc.). On peut donc écrire directement :

Workbooks('Budget.xlsm').Worksheets('Feuille 1').Range('A1')

- Plus généralement, on peut (souvent) ne pas écrire l'objet conteneur en amont, lorsqu'on sait que ce dernier est actif. Par exemple, si au cours du programme, on sait qu'on se situe forcément dans le classeur *Budget.xlsm* (qui est donc actif), il sera inutile de le mentionner dans le code. On pourra donc écrire simplement :

Worksheets('Feuille 1').Range('A1')

2.3.3. Propriétés et méthodes des objets

Lorsqu'on accède à un objet, 2 types d'actions peuvent lui être affectés :

- exploiter ses **propriétés**
- spécifier la **méthode** à utiliser par rapport à cet objet (à condition que la méthode soit compatible avec l'objet)

2.3.3.1. Propriétés d'un objet

❖ Définitions et exemples

Ces propriétés correspondent aux **paramètres** (ou **états**) de l'objet.

Exemples :

- La case A1 contient le nombre 29 \Rightarrow La propriété **Valeur** de l'objet Worksheets("Feuille 1").Range("A1") est donc une **valeur numérique** égale à 29
- La case B1 contient le nom Département \Rightarrow La propriété **Valeur** de l'objet Worksheets("Feuille 1").Range("A1") est donc **une chaîne de caractères** égale « Département »

❖ 4 Types de valeurs de propriétés :

- **Valeur numérique** \rightarrow ex. : 0 ; 3,1415 ; - 55 ; 3E4 (= 3×10^4)
 - Byte (nombre entier entre 0 et 255)
 - Integer (nombre entier entre - 32 768 et 32 767)
 - Single et Double (nombre à virgule flottante)
 - Currency (nombre à virgule fixe avec 15 chiffres pour la partie entière et 4 pour la partie décimale)
- **Chaîne de caractères** \rightarrow ex. : 'Département', 'Paris', ' il pleut', '747' (en tant que chaîne, lorsqu'on parle par exemple du modèle 747 de Boeing)
- **Valeur booléenne** (*Boolean*) qui ne prend que 2 états : **False** (ou -1) / **True** (ou 1)
- **Constante** : il s'agit d'une valeur nommée qui ne change pas au cours de l'exécution du programme et qui peut être définie (déclarée par celui qui conçoit le programme au début du programme) ou prédéfinie par VBA.

a) **Constante déclarée** : *Const*

Exemple : au cours d'un programme, on utilise dans les calculs la valeur du taux d'intérêt et la valeur de la somme placée pour calculer à différentes périodes la valeur acquise par cette somme placée à un certain taux d'intérêt.

ex. $R1 = 1000 \times (1 + 0,05)^1$; $R2 = 1000 \times (1 + 0,05)^2$ etc.

→ Au lieu d'écrire dans le programme, pour chaque calcul, la valeur numérique correspondant au taux d'intérêt (5 %) et la valeur numérique correspondant au capital (1000), ce qui est fastidieux, source d'erreur et peu productif (car il faut ré-écrire tous les codes concernés si on change la valeur du taux d'intérêt ou du capital), on va **déclarer** les **constantes** *taux_interet* et *Capital* de la manière suivante :

Const *taux_interet* as Single = 0.05 (Single = valeur réelle)

Const *Capital* as Single = 1000

→ Ces constantes seront ensuite appelées lors des différents calculs tels que :

$R1 = \text{Capital} \times (1 + \text{taux_interet})$

→ De la même manière, on peut également déclarer des constantes de types **chaîne** etc.

Exemple : Const *Musee* As String = "Le Louvre" (String = chaîne de caractères)

b) **Constantes prédéfinies par VBA** (inutile des les déclarer) :

Exemple : pour définir l'orientation de la feuille active, on peut utiliser la constante *xlLandscape* (mode paysage)

`Workbooks('Budget.xlsm').Worksheets('Feuille 1').PageSetup.Orientation = xlLandscape`

Remarque : la partie *.PageSetup.Orientation* correspond à la **méthode** qu'on assigne à l'objet *Workbooks('Budget.xlsm').Worksheets('Feuille 1')* (cf. infra).

! Si un objet est amené à prendre **plusieurs types de valeurs** (ex. *String* puis *Integer*), il faut le spécifier via **As Variant** ⇒ ex. Dim objet_variable As Variant

❖ Actions sur les propriétés : 2 types d'actions

Concernant les propriétés d'un objet, on peut envisager **2 types d'actions** :

- Examiner les paramètres actuels des propriétés de l'objet

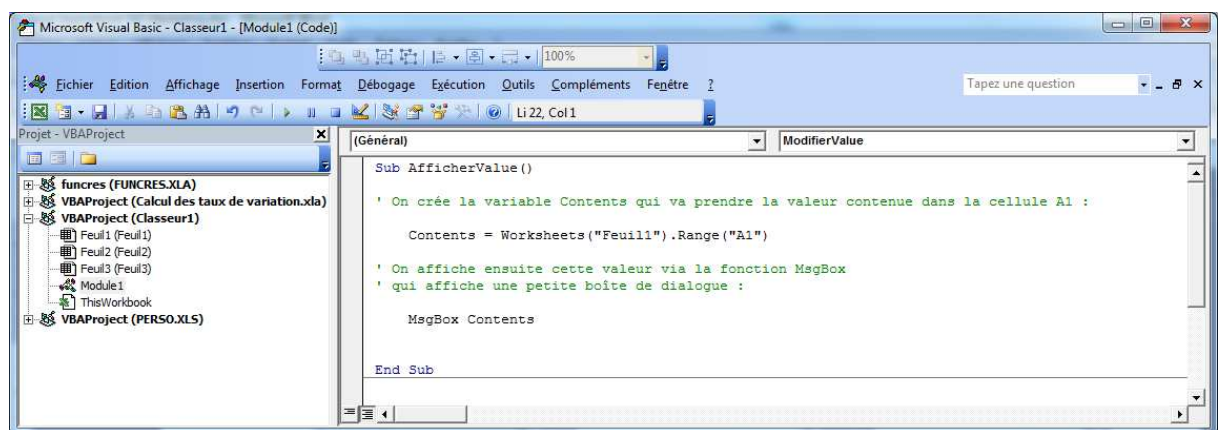
- Modifier ces paramètres de propriétés

i. **Examiner les paramètres actuels des propriétés de l'objet :**

On cherche à connaître, à un moment donné dans l'exécution d'un programme les spécificités de l'objet sélectionné.

Exemple :

- Ouvrir un nouveau classeur
- Ecrire dans la cellule A1 de la feuille 1, la valeur 235
- Ecrire à partir de l'éditeur VBE, dans le module 1 (Insertion > Module), la macro (procédure) *AfficherValue* suivante :



- Exécuter la macro *AfficherValue*
- La boîte suivante devrait apparaître :



- Inscrire dans A1 « Marketing » et exécuter de nouveau la macro
- La macro affiche aussi bien une chaîne de caractères que des nombres

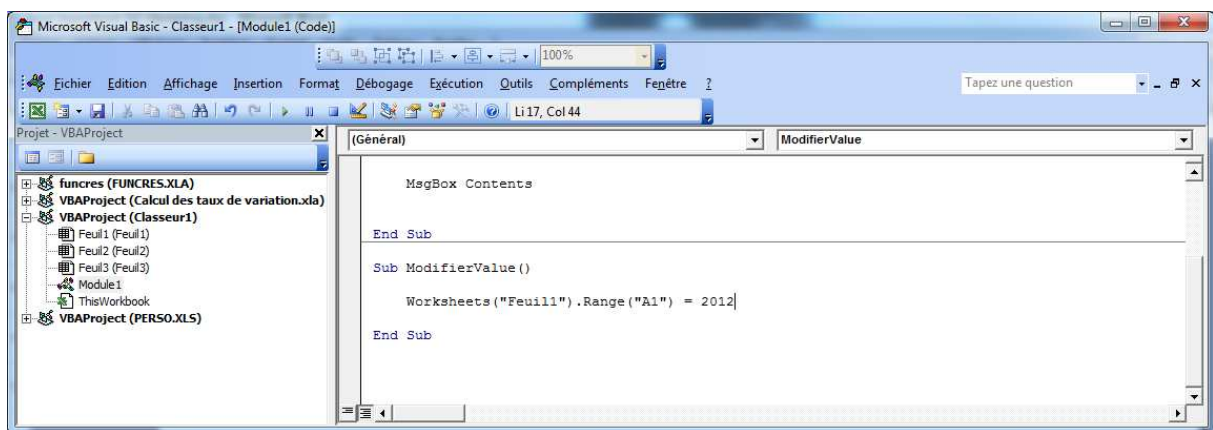
Remarque : La fonction **MsgBox** est très utile lors de la conception d'un programme VBA, car elle permet de contrôler l'exécution de VBA aux endroits jugés stratégiques dans le programme.

ii. **Modifier les valeurs attachées aux propriétés de l'objet**

Ex. modifier la valeur numérique, la chaîne de caractère, passer du statut *False* au statut *True* etc.

- Les propriétés qui peuvent être modifiées sont dites en « **lecture-écriture** »
- Les propriétés qui ne peuvent être modifiées sont dites en « **lecture seule** » (ex. la date de création du classeur)

Exemple : au lieu d'afficher le contenu de la cellule A1, on va créer une macro *ModifierValue* qui en modifie le contenu en y inscrivant le nombre « 2012 ».



Remarque : comparaison des codes entre les deux macros :

Examen : Contents = Worksheets("Feuil1").Range("A1")

Modification : Worksheets("Feuil1").Range("A1") = 2012

On remarque donc que :

- à gauche du signe égal : ce qui va être modifié
- à droite du signe égal : provenance de la valeur

2.3.3.2. **Méthodes d'un objet**

Définition :

- Une **méthode** correspond à l'**action (comportement)** qu'un objet peut exécuter (en se rappelant qu'à l'inverse, une **propriété** correspond à l'**état** d'un objet).

- Les méthodes pouvant être appliquées dépendent avant tout de l'objet :
 - certaines méthodes ne peuvent pas être exécutées par certains objets
 - certaines méthodes sont communes à plusieurs objets différents
- Une méthode peut avoir des conséquences sur l'état de certaines **propriétés** de l'objet sur lequel elle s'applique voire sur d'autres objets.

Exemple : si une méthode consiste à modifier le contenu de l'objet cellule (ex. A1), alors :

- la propriété de cette cellule A1 sera modifiée
- la propriété d'autres cellules (donc autres objets) liées par une formule à A1 (ex. B1 et C1)

- La **syntaxe** générale permettant d'appliquer une méthode est la suivante :

élément_ciblé.Méthode

où élément_ciblé : **objet ou collection d'objets** sur lequel s'applique la méthode

Exemples :

- On cherche à effacer le contenu et formatage de la plage de cellules A1 : C4 de la feuille 1
 - **objet** : plage de cellules A1 : C4 de la feuille 1 (du classeur actif)
 - **Méthode** : effacer

→ **Syntaxe** :

```
Sub Clear_Range()

    Worksheets("Feuil1").Range("A1:C4").Clear

End Sub
```

(Remarque : si on veut juste effacer le contenu de la plage sans effacer la mise en forme, la méthode sera : ClearContents).

- On cherche à copier dans la cellule B1, le contenu de la cellule A1, ces cellules étant situées dans la feuille 1

→ **Syntaxe** :


```
Sub Copy_Range()  
  
    Worksheets("Feuil1").Activate  
    Range("A1").Copy Range("B1")  
  
End Sub
```

Commentaires : en fait, on utilise **2 méthodes consécutives**

- Afin d'éviter une écriture laborieuse, on a tout d'abord **activé** la feuille 1 (méthode 1) via la **méthode Activate** : il sera alors inutile d'écrire Worksheets("Feuil1"). dans les lignes de codes suivantes.
- On applique ensuite la **méthode Copy** sur la cellule A1 (méthode 2), car c'est bien A1 qui est copié et non pas B1. On remarque en particulier que dans ce cas, la méthode Copy est suivi **d'un argument** (séparé par un espace) à savoir Range("B1") sur lequel porte la modification.

2.4. La structuration de la programmation VBA

2.4.1. Procédures et instructions

Un **programme** complet sous VBA regroupe un ensemble de **procédures** associées à des objets distincts et qui interagissent entre elles.

2.4.1.1. Procédure : c'est une séquence d'instructions codées qui s'exécutent en un bloc.

- **A l'intérieur d'un même module**, les codes sont donc regroupés en différentes procédures.
- En toute logique, chaque procédure prend en charge **une tâche spécifique** du programme :

⇒ elles sont donc relativement simples

⇒ elles rendent le programme plus performant et plus lisible (les procédures sont séparées les unes des autres par un trait horizontal).

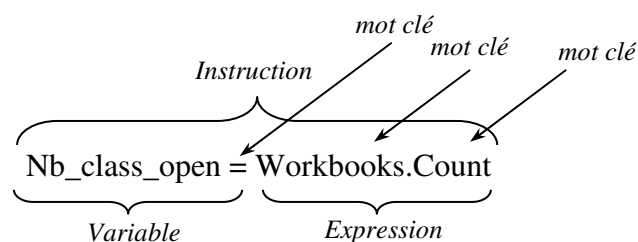
- **Procédures événementielles** : elles sont déclenchées par un **événement généré par l'utilisateur** tel que le clic sur un bouton OK ou la frappe d'une touche ⇒ tant que le clic n'a pas eu lieu, la procédure ne se déclenche pas.

2.4.1.2. Les instructions :

- ce sont des **tâches précises** codées en VBA
- les instructions sont composées de :
 - **mots clés** = vocabulaire spécifique au langage VBA pouvant être une fonction, une propriété, une méthode, un opérateur arithmétique
 - **constantes** (ne changent pas de valeur au cours de l'exécution du programme)
 - **variables** (changent de valeur au cours de l'exécution du programme)

⇒ la combinaison de ces 3 types d'éléments forment une **expression** dont la finalité est d'exécuter un tâche ou vérifier des données

Exemple :



⇒ Cette instruction donne comme valeur à la variable Nb_class_open le nombre de classeurs ouverts

- **Il existe 3 types d'instructions :**

- i. les instructions de **déclaration**

Elles servent à **nommer** (créer) un élément tel qu'une variable, une constante ou une procédure en précisant éventuellement son type (ex. nombre entier ou réel pour une variable). Ce nom sera ensuite utilisé pour mobiliser cet élément dans le projet.

Elles demeurent **invisibles** pour l'utilisateur du programme (qui n'accède pas aux codes sous VBE), étant donné qu'il ne s'agit que de déclarations et non d'actions à proprement parler.

Exemples :

Dim Nb_class_open as Integer → on déclare **la variable** Nb_class_open comme un nombre entier

Sub Certain_Click() → on déclare **la procédure** Certain_click()

End Sub → on signale la fin de la procédure

- ii. les instructions d'**affectation**

Elles affectent une valeur ou une expression à une variable, à une constante ou à une propriété.

Elles contiennent toujours l'opérateur « = »

Attention :

l'affectation via l'expression « = » ne doit donc pas être confondue avec l'égalité mathématique « = ». Par exemple :

- sous VBA, on pourra écrire, dans le cadre d'une boucle « i = i + 1 » : ceci signifie que l'on donne pour instruction d'incrémenter de 1 unité la variable de comptage i ... ainsi si i avait pour valeur initiale 0 elle prend maintenant la valeur 1, puis la valeur 2 à la boucle suivante etc (voir plus loin pour les notions de boucle).
- alors que mathématiquement, « i=i+1 » n'a aucun sens puisque cette écriture signifie que 0 = 1
Sous R (ou d'autres langages), afin d'éviter cette confusion, l'affectation s'écrit de manière différent : $i \leftarrow i+1$

Les affectations peuvent être **visibles** ou **invisibles** pour l'utilisateur, selon l'instruction qui est proposée.

Exemples :

1) Worksheets("Feuil1"). Range("A1").Font.Name = "Times New Roman"

⇒ **instruction visible** pour l'utilisateur car elle affecte la police *Times New Roman* à la **propriété** Name de l'**objet** Font (Police) de la cellule A1 de la feuille 1

2) Nb_class_open = Workbooks.Count

⇒ **instruction invisible** pour l'utilisateur car la variable *Nb_class_open* s'est vue attribuer comme valeur le nombre de classeurs ouverts, mais cette valeur n'apparaît ni sur une feuille Excel, ni dans une boîte de dialogue. Il faudrait une autre instruction pour que ce soit visible telle que :

Worksheets("Feuil1"). Range("C1") = Nb_class_open

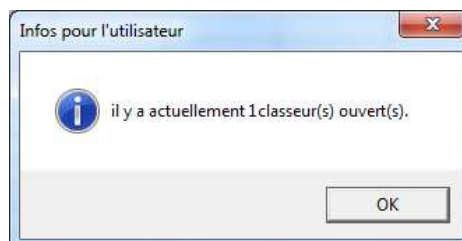
iii. les instructions **exécutables**

Ces instructions accomplissent des **actions** telles que l'**exécution** d'une **méthode** ou d'une **fonction**.

Elles sont **visibles** par l'utilisateur.

Exemple : faire afficher une **boîte de dialogue**

MsgBox "il y a actuellement " & Nb_class_open & "classeur(s) ouvert(s).", vbInformation, "Infos pour l'utilisateur"



2.4.2. Les procédures sous VBA : **Sub**, **function** et **Property**

Une **procédure** est une série d'instructions dont la finalité est d'exécuter une ou plusieurs tâches, visibles ou invisibles pour l'utilisateur.

Il existe **3 types de procédures** qui se distinguent selon le type de tâches effectuées :

- les procédures **Sub**
- les procédures **Function**
- les procédures **Property**

2.4.2.1. Les procédures **Sub** (sous-routine)

- Une procédure Sub est une série d'instructions exécutant une tâche **sans renvoyer de valeur**.
- Elle commence par **Sub** et se termine par **End Sub** :

```
Sub Nom_procedure ( )  
    Instructions  
End Sub
```

- Elle est dotée d'un **nom** (créé par le concepteur) afin de pouvoir être **invoquée** par la suite dans le programme. Ces noms doivent remplir certains critères tels que : ne pas utiliser un mot-clé réservé par VBA (ex. *Count*), commencer par une lettre, ne pas contenir les caractères @, &,\$, #, pas de points ni de points d'exclamation
- Elle peut correspondre à une **macro enregistrée** (créée sans passer par VBE).
- Elle peut contenir des **éléments optionnels** selon la syntaxe suivante :

```
Private Static Sub Nom_Procedure (Arguments)  
Public
```

- **Private / Public** : Une procédure **publique** peut être invoquée (via l'instruction *Call* Nom_Procedure) à partir de n'importe quelle procédure du programme (procédure dite « **appelante** »), même si elle n'est pas située dans le **même module**. **Par défaut** (lorsque rien n'est indiqué), la procédure sera considérée comme publique. A l'inverse, une procédure **privée** ne peut être invoquée qu'à partir d'une procédure appelante située dans le même module.
- **Static** : Lorsque la procédure se termine, les variables présentes dans cette procédure conservent les valeurs qui leurs ont été affectées au cours de la dernière utilisation de cette procédure.

Exemple : Comparer le fonctionnement du programme **avec** et **sans** *Static* devant la procédure *ProcédureAppelée()*

Dim Mavar

Sub ProcédureAppelante()

Mavar = 2

Call ProcédureAppelée()

Call ProcédureAppelée()

Call ProcédureAppelée()

End Sub

Static Sub ProcédureAppelée()

Dim MaVarStatic
MaVarStatic = MaVarStatic + Mavar
MsgBox MaVarStatic

End Sub

→ **Avec Static** : les valeurs affichées sont respectivement : 2, 4, 6

→ **Sans Static** : les valeurs affichées sont respectivement : 2, 2, 2

Explication ³ :

Avec Static attribuée à la procédure **appelée**, la **variable locale** *MaVarStatic* (**locale**, car **déclarée** à l'intérieure de la procédure appelée) conserve la valeur qu'elle a acquise à l'issue du **dernier appel** de la procédure appelée (sachant qu'elle est initialisée, lors de la 1^{ère} utilisation de cette procédure, à zéro).

Sans Static, la variable locale *MaVarStatic* ne conserve pas la valeur acquise à l'issue du **dernier appel** et se réinitialise à chaque fois à zéro.

○ **Arguments :**

- Si on affecte un **argument** (ex. arg1) à une procédure, ceci signifie que lorsque cette procédure sera invoquée (à partir de la procédure appelante), il faudra au préalable, dans la procédure appelante, affecter une valeur à cet argument, sinon **un message d'erreur** sera généré.
- En présence de **plusieurs arguments**, ces derniers sont séparés par une **virgule**
- Les arguments sont soumis à la **syntaxe optionnelle** suivante :

Optional ByVal ParamArray *Nom_argument* As type de valeur = ValeurParDéfaut
ByRef

▪ **Optional :**

Les arguments sont **facultatifs** ⇒ pas de message d'erreur si non présents dans la procédure appelante.

Les arguments **obligatoires** doivent être déclarés **avant** les arguments **facultatifs**.

³ Remarque : les noms **ProcédureAppelante** et **ProcédureAppelée** ont juste été choisis pour des raisons pédagogiques. Dans la réalité, il vaudra mieux les nommer de manière plus explicite en lien avec le programme tels que, par exemple : Sub Capital(), Static Sub Annuités etc.

Exemple : Sub Maprocédure(arg1, arg2 optional arg3, optional arg4)

- **ByVal / ByRef** : indiquent que l'argument est « **passé** » par **valeur** / par **référence (ou adresse)**

Exemple : Comparer le fonctionnement du programme **avec** et **sans** **ByVal** devant l'argument *MaVar*.

Dim MaVar

```
Sub ProcédureAppelante2()  
    MaVar = 2  
    Call ProcédureAppelée2(MaVar)  
    Call ProcédureAppelée2(MaVar)  
    Call ProcédureAppelée2(MaVar)
```

End Sub

```
Static Sub ProcédureAppelée2(ByVal MaVar)  
    Dim MaVarStatic  
    MaVarStatic = MaVarStatic + MaVar  
    MsgBox MaVarStatic  
    MaVar = 100
```

End Sub

- **Avec** **ByVal** : les valeurs affichées sont respectivement : 2, 4, 6
- **Sans** **ByVal** (ou **avec** **ByRef**) : les valeurs affichées sont respectivement : 2, 102, 202

Explication :

avec **ByVal** attribuée à l'argument *MaVar*, la variable *MaVar* est appelée uniquement **en valeur** : sa valeur (ici le nombre 2) est donc exploitée dans la procédure appelée, mais la variable *MaVar* intrinsèquement n'est pas modifiée. Par conséquent, l'instruction *MaVar = 100* située dans la procédure appelée (2), ne modifie que dans le cadre de cette procédure la valeur associée à *MaVar*, et donc de manière provisoire. Dès que l'on retourne dans la procédure appelante, *MaVar* conserve sa valeur initiale définie au début de la procédure appelante (et égale à 2).

Sans *ByVal* ou **avec *ByRef*** attribuée à l'argument *MaVar*, la variable *MaVar* est appelée **en référence (en adresse)** : c'est la variable *MaVar* *intégralement* qui est appelée et pas seulement sa valeur. Par conséquent, l'instruction *MaVar = 100* située dans la procédure appelée (2), modifie de manière effective la variable *MaVar*. Lorsqu'on retourne dans la procédure appelante, *MaVar* a donc une nouvelle valeur (égale à 100).

- **As Type de valeur** : on spécifie le **type de valeur** de l'argument qui « passe » dans la procédure (ex. *As Integer*, *As Boolean*, ...)

2.4.2.2. Les procédures **Function** (fonction)

- Une procédure **Function** est une série d'instructions exécutant une tâche en **renvoyant à une valeur** (numérique ou non) qui sera exploitée par d'autres procédures.

On peut faire le parallèle avec une fonction mathématique simple.

On définit la fonction $f(x) = x^2 + 3$

Si on invoque cette fonction *f* pour la valeur 4, cette dernière nous **renvoie** la valeur 19 :

$$f(4) = 4^2 + 3 = 19$$

Excel propose déjà un grand nombre de **fonctions prédéfinies** utilisables directement par l'utilisateur à partir de la barre de formules.

Par exemple : la **fonction Somme** (*nombre1 ; nombre2 ;...*) calcule la somme des arguments (cellules ou plages de cellules) spécifiées dans la parenthèse.

- Elle commence par **Function** et se termine par **End Function** :

```

Function Nom_procedure ( Arguments)
    Instructions
    Nom_procedure = Expression → on affecte une valeur à la fonction
End Sub

```

A la fin de son exécution, la fonction renvoie donc une valeur qui correspond à l'expression de la ligne : *Nom_procedure* = Expression

- Elle est dotée d'un **nom** (créé par le concepteur) afin de pouvoir être **invoquée** par la suite dans le programme. Ces noms doivent remplir certains critères tels que : ne pas

utiliser un mot-clé réservé par VBA (ex. *Count*), commencer par une lettre, ne pas contenir les caractères @, &,\$, #, pas de points ni de points d'exclamation

- Comme pour une procédure, elle peut contenir des **éléments optionnels** selon la syntaxe suivante :

Private Static Function Nom_Procedure (**Arguments**) *As Type de valeur*
Public

Exemple :

- On veut créer une procédure qui affiche dans une boîte de dialogue, le coût moyen à partir du coût total (CT) et de la quantité produite (q) et ce, en utilisant une **fonction de coût moyen** :

$$CM(q) = \frac{CT(q)}{q}$$

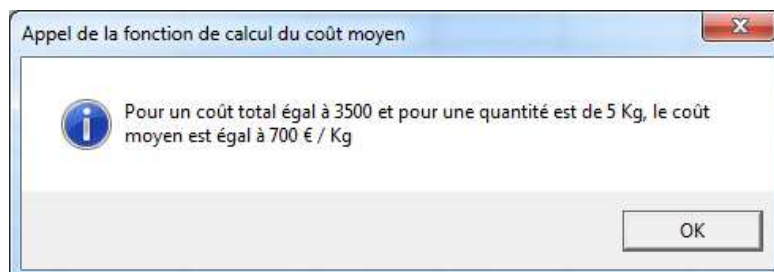
- Attention aux arguments : contrairement à l'écriture mathématique des fonctions, les fonctions sous Excel incluent comme arguments non seulement les **variables** mais aussi les **constantes**

Dans le cas présent :

D'un point de vue mathématique ou économiques : cette fonction n'a qu'un argument visible, à savoir la variable (q)

Du point de vue d'Excel : cette fonction présente 2 arguments visibles :

- la **variable** q (quantité) (ici : q = 5)
- la **constante** CT (ici : CT = 3500)
- Dans la procédure appelante, on prendra q = 10 et CT = 3500
- La boîte de dialogue désirée est la suivante :



- Les lignes de codes sont les suivantes (dans un nouveau module) :

Sub Calcul_couts() → on crée la procédure qui va utiliser la fonction CM

```
Dim q As Single
Const CT As Single = 3500
q = 5
```

```
MsgBox "Pour un coût total égal à " & CT & " et pour une quantité est de " & q & " Kg, le coût moyen est égal à " & CM(q, CT) & " € / Kg", vbInformation, "Appel de la fonction de calcul du coût moyen"
```

```
End Sub
```

Function CM(q, CT) → on crée la fonction CM qui a deux arguments

CM = CT / q

```
End Function
```

- **Remarque :** la **fonction CM** est **directement utilisable** dans le classeur qui lui est associé : tapez par exemple dans la cellule A1 de la feuille 1 = CM(10 ; 10000) ⇒ le résultat affiché dans la cellule donne alors 10 000 / 10 = 1000 (attention au séparateur d'arguments : c'est un **point-virgule** alors que dans le code VBA c'est une virgule). Pour rendre cette fonction **accessible dans d'autres classeurs**, il faut l'enregistrer comme une **macro complémentaire** (cf. instructions en amont, dans la partie 1, concernant les macros).

2.4.2.3. Les procédures **Property** (propriété) : non détaillées dans ce cours

2.5. Codage des déplacements par références absolues et relatives sous VBA

La différence entre références absolues et relatives a été déjà abordée une première fois lors de la découverte des macros.

Il convient alors d'observer quels sont les codes correspondant à ces deux types de référence.

2.5.1. Les codes associés à la position et aux déplacements

Les objets :

- **Range** : cellule, groupe de cellules qu'elles soient contigües (plages) ou non
- **Cells** : collection de toutes les cellules des classeurs actifs.
- **Row** : renvoie un numéro représentant une ligne de la feuille ⇒ **Row_s** : renvoie un objet Range (c'est-à-dire un ensemble de cellules) qui correspond à une collection de lignes
- **Column** : renvoie un numéro représentant une colonne de la feuille ⇒ **Column_s** : renvoie un objet Range qui correspond à une collection de colonnes
- **ActiveCell** : renvoie un objet Range qui représente la cellule active d'une feuille de calcul
- **Selection** : renvoie un objet Range l'objet (qui peut être une cellule, une plage de cellules,...) sélectionné dans la feuille active. Par exemple si on vient de sélectionner la cellule A1, Selection renvoie à cette cellule A1.
- **Offset** : renvoie un objet Range qui représente une plage de cellules décalée par rapport à la plage sélectionnée.

Les méthodes :

- **Select et Goto** : sélectionnent l'objet spécifié
- **Activate** : Active l'objet spécifié
- **Resize** : modifie la taille d'une plage de cellules

2.5.2. Codage des déplacements par références absolues

Astuce : Pour connaître les codes des différents déplacements, il suffit souvent de créer une macro pour chacun de ces déplacements puis d'observer les codes VBA correspondants.

- **Sélection d'une cellule**

Exemple : sélection de la cellule **C3**

⇒ Codes : `Range("C3").Select`

- **Sélection de cellules contigües puis non contigües**

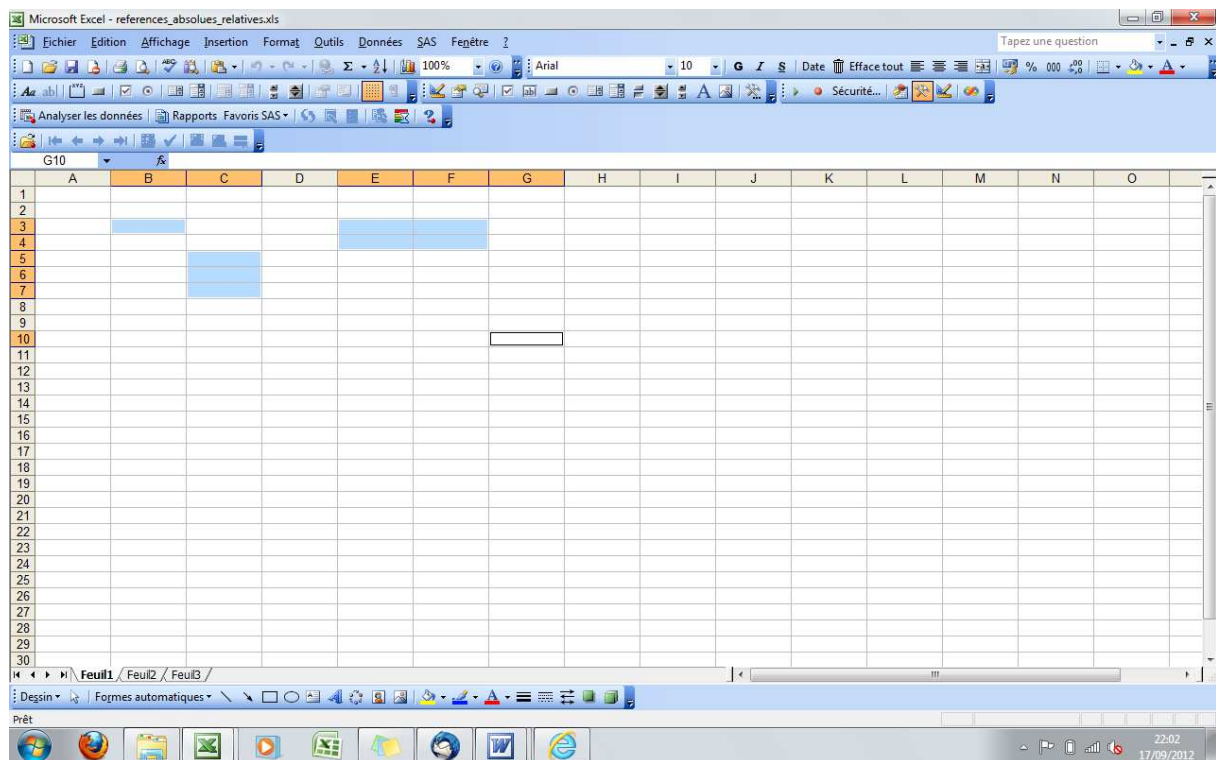
- Sélection d'une plage de cellules contigües

Exemple : plage (C4 ; E9)

⇒ Codes : `Range("C4:E9").Select`

- Sélection d'une plage de cellules non contigües

Sur la feuille Excel, la sélection de plages non contigües se fait via la **touche Ctrl** maintenues enfoncée pendant la sélection de différentes plages.



⇒ Codes : `Range("B3,C5:C7,E3:F4,G10").Select`

⇒ La séparation des plages se fait donc **via une virgule**.

- Sélections de cellules ou plages de cellules via la propriété **Cells**⁴

La propriété **Cells** permet également de coder une référence absolue à une cellule.

Sa syntaxe est la suivante :

Cells (N° ligne , N° Colonne)

où N° ligne et N° Colonne sont des coordonnées (= chiffres).

On remarque donc que cette syntaxe est bien différente de celle de Range. Par exemple, on veut écrire des mots dans la **cellule B6** :

➤ **via Range :**

```
Range("B6").Select
ActiveCell.FormulaR1C1 = "j'écis via Range"
```

➤ **via Cells :**

```
Cells(6,2).Select           → 6ème ligne, 2ème colonne
ActiveCell.FormulaR1C1 = "j'écis via Cells"
```

⇒ **avantage de Cells** : on peut faire référence aux cellules via des variables numériques, ce qui peut être intéressant dans des programmes où on cherche, par exemple, à incrémenter des cellules (ex. Cell(i + 1, j + 1))

- Sélection d'une colonne, d'une ligne

▪ **Lignes :**

Exemples :

```
lignes 6 :                ⇒ Rows("6:6").Select
lignes 6 et 7 (contigües) ⇒ Rows("6:7").Select
ligne 6 et 8 (non contigües) ⇒ Range("11:11,14:14").Select
```

▪ **Colonnes :**

Exemples :

```
Colonne E :                ⇒ Columns("E:E").Select
Colonnes D et E            ⇒ Columns("D:E").Select
```

⁴ ! La propriété Cells n'apparaît pas de manière naturelle dans les macros

Colonnes D et F (non contiguës) ⇒ **Range**("D:D,F:F").Select

Exemple : plage (C4 ; E9)

⇒ Codes : **Range**("C4:E9").Select

2.5.3. Codage des déplacements par **références relatives**

Rappel : pour les **références relatives**, ce n'est pas la position des cellules dans la feuille Excel qui importe (ex : la plage (B3 ; C3)) mais la position des cellules les unes par rapport aux autres (ex. deux cellules l'une à côté de l'autre) et plus précisément la position des cellules choisies par rapport à la **cellule active initiale**.

Exemple : si on se situe initialement en B1 (**cellule active**), et qu'on sélectionne la plage (B3 ; C3), c'est la plage horizontale de 2 cellules située **en dessous** et **à droite** de la cellule initiale qui importe.

(On utilise les mêmes exemples que pour les références relatives.)

- **Sélection d'une cellule**

Exemples :

a) sélection de la cellule **C3**, sachant que la cellule active initiale était **B1**

On observe que C3 se situe à:

- **1 colonne à droite** de la cellule de référence B1
- **2 lignes en dessous** de la cellule de référence B1

⇒ Codes : **ActiveCell.Offset**(2, 1).**Range**("A1").Select

Range("A1") : Il s'agit d'une plage d'une **seule cellule**. Comme, il s'agit d'une **référence relative**, Excel prend comme point d'ancrage, la **cellule A1 (adresse virtuelle)**.

ActiveCell.Offset(1, -3) : la cellule **sélectionnée** se situe à :

- 1 colonne à droite** de la cellule de référence
- 3 lignes en dessous** de la cellule de référence

b) sélection de la cellule **C2**, sachant que la cellule active initiale était **D4**

On observe que C2 se situe à:

- **1 colonne à gauche** de la cellule de référence

➤ 2 lignes au **dessus** de la cellule de référence)

⇒ **Codes** : `ActiveCell.Offset(-2, -1).Range("A1").Select`

- **Sélection de cellules contigües puis non contigües**

- Sélection d'une plage de cellules contigües

Exemple : sélection de la plage (C4 ; E9) sachant que la cellule active était B1

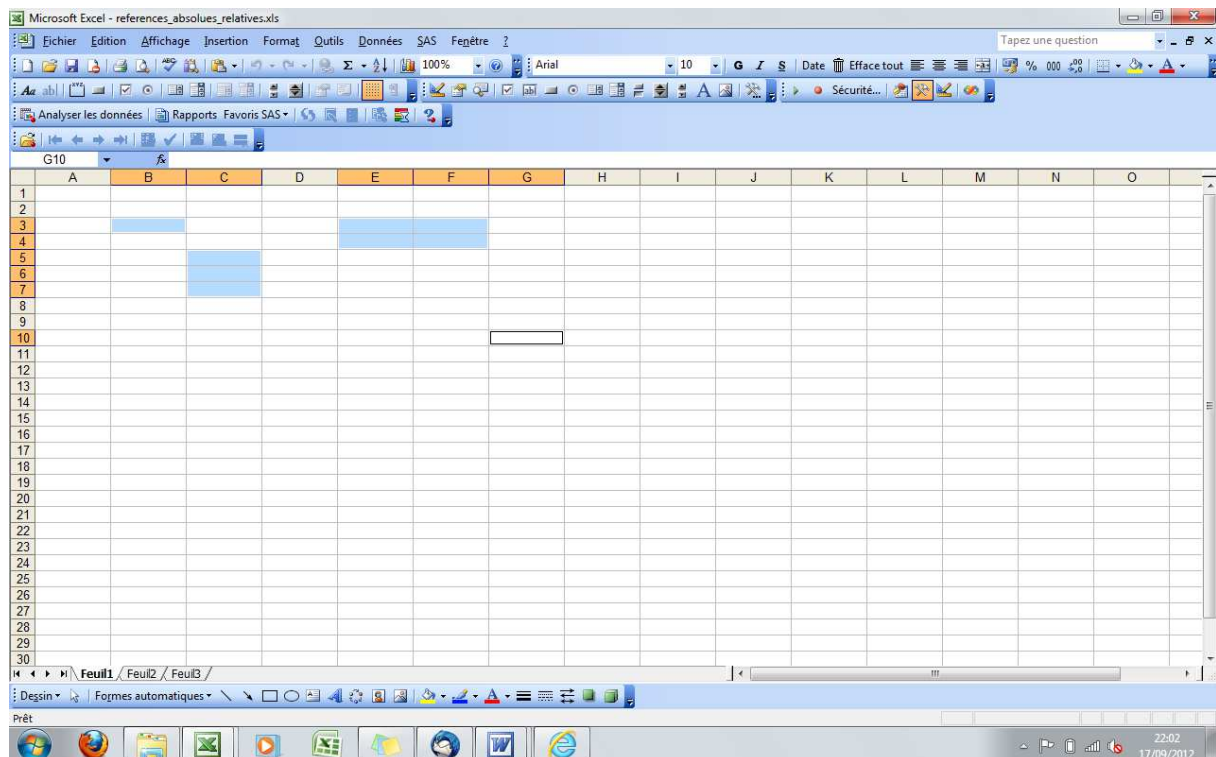
C'est une plage de 3 colonnes × 5 lignes = 15 cellules, dont **le coin supérieur gauche** (où se situe toujours la cellule active finale) se situe à :

1 **colonne** à **droite** de la cellule de référence
3 **lignes** en **dessous** de la cellule de référence

⇒ **Codes** : `ActiveCell.Offset(3, 1).Range("A1:C5").Select`

- Sélection d'une plage de cellules non contigües

Sélection des plages suivantes, sachant que la cellule active initiale est B1 :



⇒ **Codes** : `ActiveCell.Offset(2, 0).Range("A1,B3:B5,D1:E2").Select`

2.6. La déclaration des tableaux sous VBA

Tableau = ensemble de **variables** ayant en **commun leur nom**, suivi d'un numéro d'index.

Exemple : on crée le Tableau **Note** comprenant 30 variables de type *Single* correspondant à la note reçue par chacun des 30 étudiants. La première variable sera référencée sous le nom **Note(1)**, la deuxième sous le nom **Note(2)**, etc.

2.6.1. Déclaration d'un tableau simple (à 1 dimension)

❖ Syntaxe : **Dim** *Nom_tableau* (*premier_Index* **to** *dernier_index*) **As** type de valeur

Exemple : on déclare le tableau *Note* pour une liste de 30 étudiants, sachant que les données du tableau seront des nombres réels.

Dim *Note* (1 to 30) **As** Single

- Le 1^{er} index est **facultatif** : par défaut il sera fixé à zéro par VBA ; pour modifier cet index par défaut et le fixer à 1, il faut déclarer dans le module : **Option Base 1**
- **As type de valeur** (ex. Single, String, Integer, ...) est également facultative

❖ Affectation d'une valeur à une case (cellule) du tableau

Exemple : on veut affecter la note 15 à l'étudiant N°29

Note(29) = 15

Remarque : Mathématiquement, dans le cas de valeurs numériques, un tableau à une dimension correspond à un **vecteur**.

2.6.2. Déclaration d'un tableau multidimensionnel

2.6.2.1. Tableau à 2 dimensions (double entrée)

Exemple : au lieu d'avoir une seule note par étudiant, on a 5 notes par étudiants correspondant par exemple à 5 matières différentes :

Dim *Note* (1 to 30, 1 to 5) **As** Single

⇒ On obtient donc un tableau de 30 lignes × 5 colonnes = 150 cellules

Pour affecter la note 11 à la 3^{ème} matière (3^{ème} colonne) du 22^{ème} étudiant (22^{ème} ligne), on écrira :

$$Note(29,3) = 15$$

Remarque : Mathématiquement, dans le cas de valeurs numériques, un tableau à une deux dimensions correspond donc à une **matrice**.

2.6.2.2. Au-delà de 2 dimensions

Bien que VBA autorise 60 dimensions, il est assez peu courant de concevoir des applications de tableaux au-delà de 2 ou 3 dimensions.

Exemple concernant la 3^{ème} dimension : concernant les notes reçus dans 5 matières par 30 étudiants, on peut introduire le temps en considérant **l'évolution des ces notes** au cours des 6 semestres de licences

Déclaration : **Dim Note (1 to 30, 1 to 5, 1 to 6) As Single**

⇒ Il s'agit donc d'un **cube** de $30 \times 5 \times 6 = 900$ cellules

Affectation de la note 15 au 29^{ème} étudiant, dans la 3^{ème} matière et au 5^{ème} semestre de licence :

$$Note(29,3,5) = 15$$

2.7. Couleur des codes :

- **en vert :** ce sont les commentaires (précédés d'un apostrophe), qui ne sont pas des éléments exécutables
- **en bleu :** ce sont les mots-clés du langage VBA tels que Sub, End Sub, ...
- **en noir :** les instructions (ex. Range("A1"). Select)
- **en rouge :** les erreurs repérées par VBE

2.8. Les opérateurs sous VBA

Opérateur arithmétique	Signification
... + ...	addition
... - ...	soustraction
... * ...	multiplication
... / ...	division
... \ ...	division entière (ne donne que la partie entière)
... Mod ...	reste de la division entière
... ^	élévation à la puissance (ex. $3^2 = 3^2 = 9$)

Opérateur relationnel	Signification
... = ...	égal à
... > ...	supérieur à
... < ...	inférieur à
... < > ...	différent de
... > = ...	supérieur ou égal à
... < = ...	inférieur ou égal à
... Like ...	identique à (pour comparer des chaînes de caractères) ⁵
... Is ...	égal à (pour comparer des variables objets)

Opérateur logique	Signification
Not ...	<i>Non</i> : Négation d'1 expression
... And ...	<i>Et</i> : Union de 2 expressions
... Or ...	<i>Ou</i> : Désunion de 2 expressions
... Xor...	<i>Ou exclusif</i> : Exclusion d'une des 2 expressions
... Eqv ...	<i>Equivalence</i> : Equivalence des 2 expressions
... Imp...	<i>Implication</i> : Implication d'une expression sur l'autre

... &...	Concaténation
----------	---------------

Exemple :

- Concaténation de **deux chaînes de caractères** en une **seule chaîne de caractères**

“Bonjour” & “Paul” \Rightarrow “Bonjour Paul”

- Concaténation de **chaînes de caractères** et d'un nombre en une **seule chaîne de caractères**

“j'ai” & 18 “ans” \Rightarrow “J'ai 18 ans ”

- Concaténation de plusieurs nombre en un **seul nombre**

29 & 260 \Rightarrow 29260

⁵ L'opérateur ... = ... peut également être appliqué pour des chaînes de caractères, mais il ne tient pas compte de la casse des caractères.

2.9. Le développement de programmes sous VBA : éléments de base

2.9.1. Les structures de contrôles

Les **structures de contrôle** sont des instructions permettant d'orienter le comportement du programme (macro) en fonction de conditions, d'informations fournies par l'utilisateur etc.

2.9.1.1. Les instructions conditionnelles

Les **instructions conditionnelles** sont la base de la programmation : elles permettent d'orienter le comportement d'une macro en fonction de l'état du programme ou du document à un moment précis.

Il existe 2 **types** de structures conditionnelles

- **If ... Then (... Else)**
- **Select Case**

a) La structure **If ... Then (... Else)**

i. Syntaxe la plus simple avec **une seule instruction**

If Condition **Then** Instruction (*une seule instruction*)

Exemple : **If** Note < 0 or >20 **Then** MsgBox "Erreur de saisie"

ii. Syntaxe simple avec **plusieurs instructions**

If Condition **Then**

 Instruction1

 Instruction2

 ...

End If

iii. Syntaxe avec **Else**

If Condition **Then**

attention : il faut aller à la ligne !

 Instruction1

 Instruction2

Else

attention : il faut aller à la ligne !

 Instruction3

 Instruction4

 ...

End If

iv. Syntaxe avec **ElseIf**

If *ConditionA* **Then**

Instruction1

Instruction2

ElseIf *ConditionB*

Instruction3

Instruction4

⇒ les instructions3 & 4 ne seront exécutées que si la condition B est vérifiée

Else

Instruction5

Instruction6

...

⇒ Aucune condition pour que les instructions 5 & 6 soient exécutées

End If

Remarque : plusieurs conditions **If ... then** peuvent être **imbriquées** les unes dans les autres

b) La structure **Select Case**

Lorsque la condition comporte de nombreux cas de figures, la structure **If ... Else** n'est pas la plus adaptée. Il vaut mieux utiliser **Select Case**.

Syntaxe :

Select Case *Expression*

Case *Valeur1*

Instructions1

⇒ Si *Expression* a la *Valeur1*, l'*instruction1* est exécutée

Case *Valeur2*

Instructions2

Case *Valeur3*

Instructions

Case Else ⇒ Si *Expression* n'a aucune des valeurs, l'*instruction_autre* est exécutée

Instructions_autre

End Select

Exemple :

Option Explicit

⇒ option qui oblige à déclarer toutes les variables (évite les fautes de frappe)

Sub Affiche_Réduction()

Dim Quantité **As** Integer

Dim Réduction **As** Variant

⇒ permet à cette valeur d'être numérique mais aussi de type Chaîne

Quantité = **InputBox**("Combien de DVD voulez-vous acheter ?") ⇒ affiche une boîte de dialogue invitant à saisir la quantité demandée

Select Case Quantité

Case 0: Réduction = 0 ⇒ si on ne va pas à la ligne après le chiffre, il faut mettre 2 points avant l'instruction

Case 0 To 4: Réduction = 10

Case 5 To 9: Réduction = 25

Case Is > = 10: Réduction = 50

Case Else: Réduction = " ?? "

⇒ Dans tous les autres cas (ex. -5), on va afficher :??

End Select

MsgBox "La réduction sur votre achat sera de " & Réduction & "%"

End Sub

2.9.1.2. Les boucles

Les boucles sont des **instructions** qui sont **répétées** jusqu'à la survenue d'un élément engendrant la fin de la répétition. Elles permettent donc d'utiliser plusieurs fois de suite une même série d'instructions sans à avoir à ré-écrire ces instructions ⇒ elles constituent donc un moyen pratique pour **automatiser des tâches répétitives**.

Il existe **4 types** de boucles

- While ... Wend
- Do ... Loop
- For ... Next
- For Each ... Next

! Pour arrêter manuellement une boucle (par exemple parce que la boucle ne semble pas s'arrêter d'elle-même car mal programmée), il faut Faire **Ctrl + Pause** puis suspendre le programme dans VBE.

a) La boucle While ... Wend

La boucle **While ... Wend** permet de répéter une série d'instruction tant que **la condition** spécifiée pour son exécution est **vérifiée**.

Syntaxe :

While *Condition*

Instructions

Wend

avec *Condition* : expression **comparant 2 valeurs** à l'aide d'un **opérateur relationnel**

Pour rappel :

Opérateur relationnel	Signification
... = ...	égal à
... > ...	supérieur à
... < ...	inférieur à
... < > ...	différent de
... > = ...	supérieur ou égal à
... < = ...	inférieur ou égal à
... Like ...	identique à (pour comparer des chaînes de caractères) ⁶
... Is ...	égal à (pour comparer des variables objets)

Exemple : A partir d'une colonne de données numériques en Francs, on veut transformer ces données en euros (en les multipliant par 6,55957), en affichant sur la colonne à droite « conversion effectuée » et ce, jusque la routine rencontre une cellule vide (signifiant la fin du tableau).

Codes :

Sub Demo_While()

While ActiveCell.Value < > Empty *⇒ Condition : tant que la cellule vide n'es pas vide*

ActiveCell.Value = ActiveCell.Value * 6.55957

ActiveCell.Offset(0, 1) = "Conversion effectuée" *⇒ Ecriture du message dans la cellule de droite*

ActiveCell.Offset(1, 0).Select *⇒ Sélection de la cellule en dessous*

Wend

End Sub

⁶ L'opérateur ... = ... peut également être appliqué pour des chaînes de caractères, mais il ne tient pas compte de la casse des caractères.

b) Les boucles **Do ... Loop** : une généralisation de **While ... Wend**

Les boucles **Do Loop** sont semblables à **While ... End** mais en élargisse l'utilisation en proposant **4 types de boucles** :

- i. **Do While ... Loop** : **tant que** la condition est respectée, la boucle s'exécute

Do While Condition

Instructions

Loop

- ii. **Do Until ... Loop** : **Jusqu'à** ce que la condition soit réalisée, la boucle s'exécute

Do Until Condition

Instructions

Loop

- iii. **Do ... Loop While** : la boucle s'exécute, puis se répète **tant que** la condition est respectée

Do

Instructions

Loop While Condition

- iv. **Do ... Loop Until** : La boucle s'exécute puis se répète, **tant que** la condition est respectée, la boucle s'exécute

Do

Instructions

Loop Until Condition

La différence entre **Do While ... Loop** et **Do ... Loop While** réside dans le fait que :

- dans **Do While ... Loop** : le test conditionnel est exécuté en 1^{er} : s'il n'est pas vrai (=False), alors les instructions de la boucle ne sont jamais exécutées.
- dans **Do ... Loop While**, on exécute les instructions puis on vérifie le test \Rightarrow au total, les instructions de la boucle sont exécutées au moins une fois

C'est la même chose pour **Do Until ... Loop** vs **Do ... Loop Until**

c) La boucle **For ... Next**

Cette boucle permet d'exécuter un certain nombre de fois, la même série d'instructions, via l'utilisation d'un **compteur** (qu'on appelle *compteur* ou *i* ou *j* etc.) que l'on incrémente via un **pas**.

Syntaxe :

For *compteur* = *x* **To** *y* **Step** *z*

Instructions

Next *compteur*

x et *y* : respectivement la première et la dernière valeur prise par le compteur

z : le pas, c'est-à-dire de combien augmente le compteur à chaque itération de la boucle

⇒ Tant que le compteur est inférieur à *y*, les instructions de la boucle sont exécutées.

Remarques :

- le **pas** du compteur peut-être négatif, ce qui conduit à utiliser un compte à rebours.
- des boucles peuvent être imbriquées les unes dans les autres ⇒ il faut alors veiller à nommer les compteurs respectifs de manière différente (ex. : compteur1, compteur 2, ...)

d) La boucle **For Each ... Next** (boucle dans une collection d'objets)

Cette boucle permet de **généraliser** une série d'instructions à **l'ensemble des objets d'une collection d'objets** (ex. classeurs, feuilles,etc.)

Syntaxe :

For Each *Element* **In** *Collection*

Instructions

Next *Element*

Exemple :

Programme qui supprime chaque ligne 1 de toutes les feuilles du classeur.

```
Sub Supprime_ligne_1()  
    Dim Feuille as Worksheet  
    For Each Feuille In ActiveWorkbook.Worksheets  
        Feuille.Rows(1).Delete  
    Next Feuille  
End Sub
```

2.9.1.3. Les instructions Goto

L'utilisation de **Goto** n'est quasiment plus utilisée actuellement : elle repose sur une numérotation préalable (1 : ... 2 :) des lignes de programme et renvoie le déroulement du programme à la ligne indiquée par Goto (ex. Goto 2).

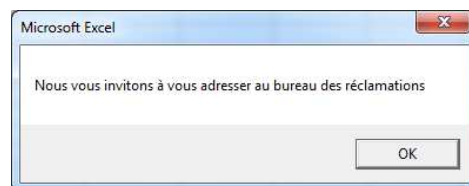
2.9.2. L'interaction avec l'utilisateur : les boîtes de dialogues

Ces boîtes de dialogues standardisées permettent une interaction entre le programme et l'utilisateur. On peut en retenir 2 :

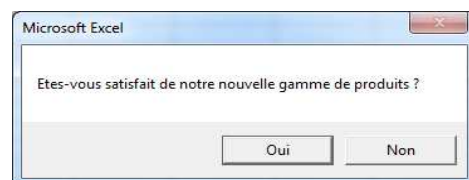
- la fonction **MsgBox**
- la fonction **InputBox**
-

2.9.2.1. La fonction MsgBox

Message simple : MsgBox "Nous vous invitons à vous adresser au bureau des réclamations"



Message exigeant un choix de la part de l'utilisateur :



Exemple :

```
Sub satisfaction()
```

```
Dim Réponse As Integer
```

```
Réponse = MsgBox("Etes-vous satisfait de notre nouvelle gamme de produits ?", vbYesNo)
```

```
Select Case Réponse
```

```
Case vbYes
```

```
MsgBox ("Nous vous invitons à vous adresser à M. Dupont")
```

```
Case vbNo
```

```
MsgBox ("Nous vous invitons à vous adresser au bureau des réclamations")
```

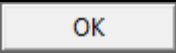
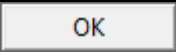
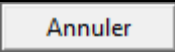
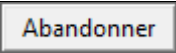
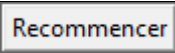

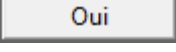
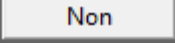
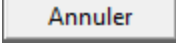
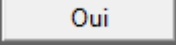
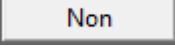
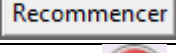
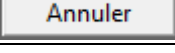




```
End Select
```

```
End Sub
```

Personnalisation de la boîte de dialogue :

Exemple : Réponse = MsgBox ("Ce questionnaire risque d'être long.", vbOKCancel + vbExclamation)

On peut ainsi combiner plusieurs éléments : vbOKCancel + vbExclamation

Constante	Valeur	Description
vbOKOnly	0	
vbOKCancel	1	 
vbAbortRetryIgnore	2	  
vbYesNoCancel	3	  
vbYesNo	4	 
vbRetryCancel	5	 
vbCritical	16	
vbQuestion	32	
vbExclamation	48	
vbInformation	64	
vbDefaultButton1	0	Bouton par défaut : Bouton 1
vbDefaultButton2	256	Bouton par défaut : Bouton 2
vbDefaultButton3	512	Bouton par défaut : Bouton 3
vbApplicationModal	0	Force l'utilisateur à répondre avant de poursuivre avec Excel
vbSystemModal	4096	Force l'utilisateur à répondre avant de poursuivre avec d'autres applications (boîte de dialogue au premier plan)

2.9.2.2. La fonction InputBox

Cette boîte de dialogue attend de l'utilisateur qu'il rentre dans l'espace prévu de la boîte l'élément demandé, que ce soit un nombre ou une chaîne de caractères.

Syntaxe :

InputBox(« Message », « Titre de la boîte », valeur par défaut)

2.7.3. Call, Exit

Graphiques

2.10. Le développement d'interfaces utilisateur (Userform, boîte de dialogue)

2.11. L'utilisation des Contrôles ActiveX

2.12. La programmation d'événements Excel

Annexes

Les différentes variables en VBA

Type de donnée	Taille d'un enregistrement	Plage
Boolean	2 octets	Vrai ou Faux.
Integer	2 octets	-32 768 à 32 767.
Long (entier long)	4 octets	-2 147 483 648 à 2 147 483 647.
Single (valeur à virgule flottante en simple précision)	4 octets	-3,402823E38 à -1,401298E-45 pour les valeurs négatives; 1,401298E-45 à 3,402823E38 pour les valeurs positives.
Double (valeur à virgule flottante en double précision)	8 octets	-1,79769313486232E308 à -4,94065645841247E-324 pour les valeurs <0; 4,94065645841247E-324 à 1,79769313486232E308 pour les valeurs >0.
Currency (monétaire)	8 octets	-922 337 203 685 477,5808 à 922 337 203 685 477,5807.
Date	8 octets	1 ^{er} janvier 100 au 31 décembre 9999.
Object	4 octets	Toute référence à des données de type Objet.
String	1 octet par caractère	0 à environ 2 milliards (environ 65 535 pour Microsoft Windows versions 3.1 et antérieures).
Decimal	14 octets	+/-79 228 162 514 264 337 593 543 950 335 sans séparateur décimal ; +/-7,9228162514264337593543950335 avec 28 chiffres à droite du séparateur décimal ; le plus petit nombre différent de zéro est +/-0.000000000000000000000000000001.
Variant	16 octets + 1 octet pour chaque car.	N'importe quelle valeur numérique dans la plage d'une valeur de type Double ou n'importe quel texte de caractères (de 0 à environ 2 milliards de car.).

Les principales instructions et procédures sous VBA

APPACTIVATE	Cette procédure permet d'activer la fenêtre d'application ayant
BEEP	Cette commande permet d'émettre un signal sonore.
CALL	Cette instruction permet d'appeler une procédure ou une fonction.
CALLBYNAME	Cette instruction permet d'associer une méthode à un objet, fixer ou demande une propriété d'un objet.
CHDIR	Cette commande permet de changer de répertoire courant
CHDRIVE	Cette commande permet de changer de l'unité de disque courante
CIRCLE	Cette procédure permet d'afficher un cercle de pixels dans l'objet.
CLOSE	Cette procédure permet de fermer un fichier «Handle» ou un périphérique.
CLS	Cette procédure permet d'effacer la zone de l'objet.
CONST	Cette instruction permet de définir une constante.
DATE	Cette instruction fixe la date courante du système
DIM	Cette commande permet de définir un ou des tableau(x) ou variable(s).
ENUM	Cette instruction permet de définir une énumération de constante.
EXIT SUB	Ces instructions permettent de quitter immédiatement une procédure.
FOR	Cette instruction permet d'effectuer un compteur en boucle.
FUNCTION	Cette instruction permet de définir une fonction.
IF	Cette instruction permet d'effectuer des vérifications conditionnel.
INPUT	Cette instruction permet la lecture d'un bloc d'octets d'un fichier «Handle».
INPUTBOX	Cette procédure permet d'afficher une boîte de dialogue avec une question et un bouton «Ok» et «Annuler».
KILL	Cette commande permet d'effacer un fichier.
LINE	Cette procédure permet d'afficher un ligne de pixels dans l'objet.
LOADPICTURE	Cette procédure permet la lecture d'un «Canvas» contenu dans un fichier de type «Bitmap».
MSGBOX	Cette procédure permet d'afficher un boîte de dialogue avec un message à l'intérieur.
ON ERROR GOTO/RESUME	Cette instruction permet d'effectuer un branchement en cas d'une erreur d'exécution.
OPEN	Cette procédure permet l'ouverture d'un fichier.

PRINT	Cette procédure permet d'afficher un texte dans l'objet ou d'écrire dans un fichier «Handle».
PRIVATE	Cette instruction permet de déclarer privé une instruction, une fonction, un objet, une variable ou une constante.
PSET	Cette procédure permet d'afficher un pixel dans l'objet.
PUBLIC	Cette instruction permet de déclarer publique une instruction, une fonction, un objet, une variable ou une constante.
RANDOMIZE	Cette instruction permet de réinitialiser les nombres aléatoires.
RESUME	Cette instruction permet de reprendre l'exécution du programme à la même ligne où ses produits l'erreur.
RESUME NEXT	Cette instruction permet de reprendre l'exécution du programme à la ligne suivante où ses produits l'erreur.
SAVEPICTURE	Cette procédure permet d'enregistrer un «Canvas» dans un fichier de type «Bitmap».
SELECT CASE	Cette instruction permet de définir une liste de conditionnels ayant comme point de comparaison un même résultat.
SUB	Cette instruction permet de définir une procédure.
TYPE	Cette instruction permet de créer une structure d'enregistrement comme les «RECORD» de Pascal ou les «struct» du langage C.

Les principales fonctions VBA

D'après le site : <http://dominiquemaniez.developpez.com/access/fonctionsVBA/>

I Les fonctions de chaîne

Asc	Renvoie une donnée de type Integer représentant le code de caractère correspondant à la première lettre d'une chaîne.
Chr	Renvoie une valeur de type String contenant le caractère associé au code de caractère indiqué.
CSTR	Renvoie une valeur de type String
InStr	Renvoie une valeur de type Variant (Long) indiquant la position de la première occurrence d'une chaîne à l'intérieur d'une autre chaîne.
InStrRev	Renvoie la position d'une occurrence d'une chaîne dans une autre, à partir de la fin de la chaîne.
LCase	Renvoie une valeur de type String convertie en minuscules.
Left	Renvoie une valeur de type Variant (String) contenant le nombre indiqué de caractères d'une chaîne en partant de la gauche.
Len	Renvoie une valeur de type Long contenant le nombre de caractères d'une chaîne ou le nombre d'octets requis pour stocker une variable.
LTrim	Renvoie une valeur de type Variant (String) contenant une copie d'une chaîne en supprimant les espaces de gauche.
Mid	Renvoie une valeur de type Variant (String) contenant un nombre indiqué de caractères extraits d'une chaîne de caractères.
Replace	Renvoie une chaîne dans laquelle une sous-chaîne spécifiée a été remplacée plusieurs fois par une autre sous-chaîne.
Right	Renvoie une valeur de type Variant (String) contenant le nombre indiqué de caractères d'une chaîne en partant de la droite.
RTrim	Renvoie une valeur de type Variant (String) contenant une copie d'une chaîne en supprimant les espaces de droite.
Space	Renvoie une valeur de type Variant (String) comprenant le nombre d'espaces indiqué.
Str	Renvoie une valeur de type Variant (String) représentant un nombre.
StrComp	Renvoie une valeur de type Variant (Integer) indiquant le résultat d'une comparaison de chaînes.
StrConv	Renvoie une valeur de type Variant (String) convertie au format indiqué.
String	Renvoie une valeur de type Variant (String) contenant une chaîne constituée d'un caractère répété sur la longueur indiquée.

StrReverse	Renvoie une chaîne contenant des caractères dont l'ordre a été inversé par rapport à une chaîne donnée.
Trim	Renvoie une valeur de type Variant (String) contenant une copie d'une chaîne en supprimant les espaces de gauche et de droite.
UCase	Renvoie une valeur de type Variant (String) contenant la chaîne indiquée, convertie en majuscules.

III. Les fonctions de date

Date	Renvoie une valeur de type Variant (Date) contenant la date système actuelle.
DateAdd	Renvoie une valeur de type Variant (Date) contenant une date à laquelle un intervalle de temps spécifié a été ajouté.
DateDiff	Renvoie une valeur de type Variant (Long) indiquant le nombre d'intervalles de temps entre deux dates données.
DatePart	Renvoie une valeur de type Variant (Integer) contenant l'élément spécifié d'une date donnée.
DateSerial	Renvoie une valeur de type Variant (Date) correspondant à une année, un mois et un jour déterminés.
DateValue	Renvoie une valeur de type Variant (Date).
Day	Renvoie une valeur de type Variant (Integer) indiquant un nombre entier compris entre 1 et 31, inclus, qui représente le jour du mois.
Hour	Renvoie une valeur de type Variant (Integer) indiquant un nombre entier compris entre 0 et 23 inclus, qui représente l'heure du jour.
Minute	Renvoie une valeur de type Variant (Integer) indiquant un nombre entier compris entre 0 et 59, inclus, qui représente la minute de l'heure en cours.
Month	Renvoie une valeur de type Variant (Integer) indiquant un nombre entier compris entre 1 et 12, inclus, qui représente le mois de l'année.
MonthName	Renvoie une chaîne indiquant le mois spécifié.
Now	Renvoie une valeur de type Variant (Date) indiquant la date et l'heure en cours fournies par la date et l'heure système de votre ordinateur.
Second	Renvoie une valeur de type Variant (Integer) indiquant un nombre entier compris entre 0 et 59, inclus, qui représente la seconde de la minute en cours.
Time	Renvoie une valeur de type Variant (Date) indiquant l'heure système en cours.
Timer	Renvoie une valeur de type Single représentant le nombre de secondes écoulées depuis minuit.
TimeSerial	Renvoie une valeur de type Variant (Date) contenant une heure précise (heure, minute et seconde).
TimeValue	Renvoie une valeur de type Variant (Date) contenant une heure.
Weekday	Renvoie une valeur de type Variant (Integer) contenant un nombre entier qui représente le jour de la semaine.
WeekdayName	Renvoie une chaîne indiquant le jour de la semaine spécifié.
Year	Renvoie une valeur de type Variant (Integer) contenant un nombre entier qui représente l'année.

IV. Les fonctions mathématiques

Abs	Renvoie une valeur de même type que celle transmise, indiquant la valeur absolue d'un nombre.
Atn	Renvoie une valeur de type Double indiquant l'arctangente d'un nombre.
Cos	Renvoie une valeur de type Double indiquant le cosinus d'un angle.

Exp	Renvoie une valeur de type Double indiquant la valeur de e (base des logarithmes népériens) élevé à une puissance.
Fix	Renvoie la partie entière d'un nombre.
Hex	Renvoie une valeur de type String représentant un nombre sous forme hexadécimale.
Int	Renvoie la partie entière d'un nombre.
Log	Renvoie une valeur de type Double indiquant le logarithme népérien d'un nombre.
Oct	Renvoie une valeur de type Variant (String) représentant la valeur octale d'un nombre.
Partition	Renvoie une chaîne de caractères de type Variant (String) indiquant l'endroit où un nombre apparaît au sein d'une série calculée de plages de valeurs.
Rnd	Renvoie une valeur de type Single contenant un nombre aléatoire.
Round	Renvoie un nombre arrondi à un nombre spécifié de positions décimales.
Sgn	Renvoie une valeur de type Variant (Integer) indiquant le signe d'un nombre.
Sin	Renvoie une valeur de type Double indiquant le sinus d'un angle.
Sqr	Renvoie une valeur de type Double indiquant la racine carrée d'un nombre.
Tan	Renvoie une valeur de type Double indiquant la tangente d'un angle.
Val	Renvoie le nombre contenu dans une chaîne de caractères sous la forme d'une valeur numérique d'un type approprié.
Abs	Renvoie une valeur de même type que celle transmise, indiquant la valeur absolue d'un nombre.
Atn	Renvoie une valeur de type Double indiquant l'arctangente d'un nombre.
Cos	Renvoie une valeur de type Double indiquant le cosinus d'un angle.
Exp	Renvoie une valeur de type Double indiquant la valeur de e (base des logarithmes népériens) élevé à une puissance.
Fix	Renvoie la partie entière d'un nombre.
Hex	Renvoie une valeur de type String représentant un nombre sous forme hexadécimale.
Int	Renvoie la partie entière d'un nombre.
Log	Renvoie une valeur de type Double indiquant le logarithme népérien d'un nombre.
Oct	Renvoie une valeur de type Variant (String) représentant la valeur octale d'un nombre.
Partition	Renvoie une chaîne de caractères de type Variant (String) indiquant l'endroit où un nombre apparaît au sein d'une série calculée de plages de valeurs.
Rnd	Renvoie une valeur de type Single contenant un nombre aléatoire.
Round	Renvoie un nombre arrondi à un nombre spécifié de positions décimales.
Sgn	Renvoie une valeur de type Variant (Integer) indiquant le signe d'un nombre.
Sin	Renvoie une valeur de type Double indiquant le sinus d'un angle.
Sqr	Renvoie une valeur de type Double indiquant la racine carrée d'un nombre.
Tan	Renvoie une valeur de type Double indiquant la tangente d'un angle.
Val	Renvoie le nombre contenu dans une chaîne de caractère sous la forme d'une valeur numérique d'un type approprié.

V. Les fonctions financières

Ces fonctions sont l'équivalent des fonctions financières que l'on trouve dans Excel (la troisième colonne du tableau indique d'ailleurs le nom correspondant Excel). Si vous développez une macro avec Excel, il est sans doute préférable d'utiliser les fonctions internes d'Excel mais si vous devez faire du calcul financier dans un programme Word ou Access, il sera plus facile d'utiliser les fonctions de Visual Basic.

DDB	Renvoie une valeur de type Double indiquant l'amortissement d'un bien au cours d'une période spécifique en utilisant la méthode d'amortissement dégressif à taux double ou toute autre méthode précisée.	DDB
FV	Renvoie une valeur de type Double indiquant le futur montant d'une annuité basée sur des versements constants et périodiques et sur un taux d'intérêt fixe.	VC
IPmt	Renvoie une valeur de type Double indiquant le montant, sur une période donnée, d'une annuité basée sur des versements constants et périodiques et sur un taux d'intérêt fixe.	INTPER
IRR	Renvoie une valeur de type Double indiquant le taux de rendement interne d'une série de mouvements de trésorerie périodiques (paiements et encaissements).	TRI
MIRR	Renvoie une valeur de type Double indiquant le taux de rendement interne modifié d'une série de mouvements de trésorerie périodiques (paiements et encaissements).	TRIM
Nper	Renvoie une valeur de type Double indiquant le nombre d'échéances d'une annuité basée sur des versements constants et périodiques et sur un taux d'intérêt fixe.	NPM
NPV	Renvoie une valeur de type Double indiquant la valeur nette actuelle d'un investissement, calculée en fonction d'une série de mouvements de trésorerie périodiques (paiements et encaissements) et d'un taux d'escompte.	VAN
Pmt	Renvoie une valeur de type Double indiquant le montant d'une annuité basée sur des versements constants et périodiques et sur un taux d'intérêt fixe.	VPM
PPmt	Renvoie une valeur de type Double indiquant le remboursement du capital, pour une échéance donnée, d'une annuité basée sur des versements constants et périodiques et sur un taux d'intérêt fixe.	PRINCPER
PV	Renvoie une valeur de type Double indiquant le montant actuel d'une annuité basée sur des échéances futures constantes et périodiques, et sur un taux d'intérêt fixe.	VA
Rate	Renvoie une valeur de type Double indiquant le taux d'intérêt par échéance pour une annuité.	TAUX
SLN	Renvoie une valeur de type Double indiquant l'amortissement linéaire d'un bien sur une période donnée.	AMORLIN
SYD	Renvoie une valeur de type Double indiquant l'amortissement global d'un bien sur une période donnée.	SYD

VI. Les fonctions de gestion de fichiers

Ces fonctions permettent de manipuler des fichiers au sens large du terme et vous pouvez aussi bien obtenir le nom d'un fichier, connaître ses attributs, calculer sa longueur que le parcourir octet par octet. Cela étant, on préférera souvent une autre méthode pour manipuler des fichiers et notamment par le biais de l'objet **FileSystemObject**.

CurDir	Renvoie une valeur de type Variant (String) indiquant le chemin en cours.
Dir	Renvoie une valeur de type String représentant le nom d'un fichier, d'un répertoire ou d'un dossier correspondant à une chaîne de recherche, à un attribut de fichier ou au nom de volume d'un lecteur.
EOF	Renvoie une valeur de type Integer contenant la valeur Boolean True lorsque la fin d'un fichier ouvert en mode Random ou Input séquentiel est atteinte.
FileAttr	Renvoie une valeur de type Long représentant le mode d'ouverture des fichiers avec l'instruction Open.
FileDateTime	Renvoie une valeur de type Variant (Date) indiquant la date et l'heure de création ou de dernière modification d'un fichier.
FileLen	Renvoie une valeur de type Long indiquant la longueur en octets d'un fichier.

FreeFile	Renvoie une valeur de type Integer représentant le prochain numéro de fichier pouvant être utilisé par l'instruction Open.
GetAttr	Renvoie une valeur de type Integer indiquant les attributs du fichier ou du dossier.
Input	Renvoie une valeur de type String contenant les caractères lus dans un fichier ouvert en mode Input ou Binary.
Loc	Renvoie une valeur de type Long indiquant la position de lecture/écriture courante dans un fichier ouvert.
LOF	Renvoie une valeur de type Long représentant la taille, exprimée en octets, d'un fichier ouvert à l'aide de l'instruction Open.
Seek	Renvoie une valeur de type Long indiquant la position de lecture/écriture courante dans un fichier ouvert à l'aide de l'instruction Open.

VII. Les fonctions logiques

Choose	Sélectionne et renvoie une valeur à partir d'une liste d'arguments.
Iif	Renvoie l'un ou l'autre de deux arguments selon l'évaluation d'une expression.
IsDate	Renvoie une valeur de type Boolean qui indique si une expression peut être convertie en date.
IsEmpty	Renvoie une valeur de type Boolean indiquant si une variable a été initialisée.
IsError	Renvoie une valeur de type Boolean qui indique si une expression est une valeur d'erreur.
IsMissing	Renvoie une valeur de type Boolean qui indique si un argument facultatif de type Variant a été passé dans une procédure.
IsNull	Renvoie une valeur de type Boolean qui indique si une expression ne contient aucune donnée valide (Null).
IsNumeric	Renvoie une valeur de type Boolean qui indique si une expression peut être interprétée comme un nombre.
IsObject	Renvoie une valeur de type Boolean qui indique si un identificateur représente une variable objet.
Switch	Évalue une liste d'expressions et renvoie une valeur de type Variant ou une expression associée à la première expression de la liste qui a pour valeur True.

Parmi toutes ces fonctions, vous utiliserez surtout la fonction **Iif** qui permet de faire un test conditionnel **If Then** sur une seule ligne et les fonctions **IsEmpty** et **IsNull**.

Exemple :

```
Dim sexe as string
sexe=IIF(civilite="Mr","Masculin","Féminin")
```

VIII. Les fonctions de conversion

Les fonctions de conversion sont extrêmement importantes et elles seront bien souvent le seul remède pour ne pas commettre des erreurs de type.

Vous devez vous rappeler qu'il est en effet impossible de mélanger les types de données au sein d'une même expression ; si je souhaite, par exemple, afficher dans une boîte de dialogue le nom d'un étudiant suivi de sa moyenne générale aux examens, il faut que je fasse une conversion de la note puisque je n'ai pas le droit de mettre bout à bout une variable caractère et une variable numérique. Je vais donc employer une fonction de conversion qui va changer le type de la variable numérique en caractère.

Le programme suivant illustre cette technique :

```

Dim nometudiant As String
Dim moyenne As Double
nometudiant = "MARTIN"
moyenne = 12
' Erreur de type
' MsgBox (nometudiant + " : " + moyenne)
' Conversion donc pas d'erreur
MsgBox (nometudiant + " : " + CStr(moyenne))

```

Les fonctions de conversion convertissent une expression en un type de données spécifique. Leur syntaxe est la suivante :

Nom_de_fonction(expression)

L'argument *expression* peut être n'importe quelle expression de chaîne ou expression numérique et le nom de la fonction détermine le type renvoyé, comme le montre le tableau suivant :

Fonction	Type renvoyé	Plage de valeurs de l'argument expression
CBool	Boolean	Toute chaîne ou expression numérique valide.
CByte	Byte	0 à 255.
CCur	Currency	-922 337 203 685 477,5808 à 922 337 203 685 477,5807.
CDate	Date	Toute expression de date valide.
CDbl	Double	-1.79769313486231E308 à

Si l'argument *expression* passé à la fonction excède la plage de valeurs du type de données cible, une erreur se produit. Il est donc préférable avant de réaliser une conversion de s'assurer qu'elle soit valide. Vous utiliserez pour ce faire des fonctions logiques comme par exemple la fonction **IsDate** pour déterminer si la valeur de l'argument date peut être convertie en date ou en heure.

La fonction **CDate** reconnaît les littéraux date et heure ainsi que certains nombres appartenant à la plage de dates autorisées. Lors de la conversion d'un nombre en date, la partie entière du nombre est convertie en date. Si le nombre comprend une partie décimale, celle-ci est convertie en heures exprimées en partant de minuit.

La fonction **CDate** reconnaît les formats de date définis dans les paramètres régionaux de votre système. L'ordre des jours, mois et années risque de ne pouvoir être défini si les données sont fournies dans un format différent des paramètres de date reconnus. De plus, les formats de date complets précisant le jour de la semaine ne sont pas reconnus. Il existe d'autres fonctions de conversion comme **Str** ou **Val** mais il vaut mieux utiliser les fonctions normalisées. Par exemple, la fonction **Val** ne respecte pas les conventions étrangères alors que la fonction **CCur** reconnaît divers types de séparateurs décimaux, de séparateurs des milliers et diverses options monétaires, selon les paramètres régionaux de votre ordinateur.

IX. Les fonctions système

Ces fonctions donnent des renseignements sur l'état du système et vous en aurez rarement besoin car certaines sont vraiment très spécialisées. La fonction **VarType** peut être intéressante pour bien comprendre le rôle des types de données.

Command	Renvoie la partie argument de la ligne de commande utilisée pour lancer Microsoft Visual Basic ou un programme exécutable développé avec Visual Basic.
DoEvents	Arrête momentanément l'exécution afin que le système d'exploitation puisse traiter d'autres événements.
Environ	Renvoie la valeur de type String associée à une variable d'environnement du système d'exploitation. Non disponible sur le Macintosh.
GetAllSettings	Renvoie une liste des clés et leurs valeurs respectives (créées à l'origine à l'aide de l'instruction SaveSetting), figurant dans une entrée d'application de la base de registres de Windows..
GetSetting	Renvoie une valeur de clé d'une entrée d'application de la base de registres de Windows.
IMEStatus	Renvoie une valeur de type Integer indiquant le mode IME (Input Method Editor) en cours de Microsoft Windows ; disponible uniquement dans les versions destinées aux pays asiatiques.
MacID	Utilisée sur Macintosh pour convertir une constante à quatre caractères en une valeur pouvant être exploitée par les fonctions Dir, Kill, Shell et AppActivate.
MacScript	Exécute un script AppleScript et retourne une valeur renvoyée par le script, le cas échéant.
QBColor	Renvoie une valeur de type Long indiquant le code de couleur RGB correspondant au numéro de couleur indiqué.
RGB	Renvoie un entier de type Long représentant le code RGB.
Shell	Lance un programme exécutable et renvoie une valeur de type Variant (Double) représentant l'identificateur (ID) de la tâche exécutée en cas de succès, ou un zéro en cas d'échec.
TypeName	Renvoie une valeur de type String qui fournit des informations sur une variable.
VarType	Renvoie une valeur de type Integer qui indique le sous-type d'une variable.

X. Les fonctions de tableau

Rappel : un tableau est une variable particulière qui peut contenir plusieurs valeurs ; chaque valeur du tableau s'appelle un élément et on accède à chaque élément par un numéro nommé indice.

Array	Renvoie une variable de type Variant contenant un tableau.
Filter	Renvoie un tableau de base zéro contenant un sous-ensemble d'un tableau de chaîne basé sur des critères de filtre spécifiés.
IsArray	Renvoie une valeur de type Boolean qui indique si une variable est un tableau.
Join	Renvoie une chaîne créée par la jonction de plusieurs sous-chaînes contenues dans un tableau.
LBound	Renvoie une valeur de type Long contenant le plus petit indice disponible pour la dimension indiquée d'un tableau.
Split	Renvoie un tableau de base zéro à une dimension contenant le nombre spécifié de sous-chaînes.
UBound	Renvoie une valeur de type Long contenant le plus grand indice disponible pour la dimension indiquée d'un tableau.

XI. Les fonctions de gestion d'objet

CallByName	Exécute une méthode d'un objet ou définit ou renvoie une propriété d'un objet.
CreateObject	Crée et renvoie une référence à un objet ActiveX.
GetObject	Renvoie une référence à un objet fourni par un composant ActiveX.

XII. Les fonctions de gestion d'erreur

Les fonctions de gestion d'erreur vous permettent de traiter préventivement les erreurs éventuelles qui peuvent se produire dans votre programme.

CVErr	Renvoie une donnée de type Variant et de sous-type Error contenant un numéro d'erreur spécifié par l'utilisateur.
Error	Renvoie le message d'erreur correspondant à un numéro d'erreur donné.

XIII. Les fonctions de formatage

Format	Renvoie une valeur de type Variant (String) contenant une expression formatée en fonction des instructions contenues dans l'expression de mise en forme.
FormatCurrency	Renvoie une expression formatée sous forme de valeur de type Currency utilisant le symbole monétaire défini dans le Panneau de configuration du système.
FormatDateTime	Renvoie une expression formatée sous forme de date ou d'heure.
FormatNumber	Renvoie une expression formatée sous forme de nombre.
FormatPercent	Renvoie une expression formatée sous forme de pourcentage (multiplié par 100) avec un caractère % de fin.

XIV. Les fonctions d'interface utilisateur

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 valeur de type String.
MsgBox	Affiche un message dans une boîte de dialogue, attend que l'utilisateur clique sur un bouton, puis renvoie une valeur de type Integer qui indique le bouton choisi par l'utilisateur.

Il est très important de noter que la valeur renvoyée par **InputBox** est une variable **String** ce qui signifie que si vous faites saisir un nombre à un utilisateur, il faudra en principe le convertir, grâce à une fonction de conversion telle que **Cdbl**, **CInt** ou **CLng**, en une variable numérique.

XV. Les fonctions d'impression

Spc	Utilisée avec l'instruction Print # ou la méthode Print pour positionner la sortie.
Tab	Utilisée avec l'instruction Print # ou la méthode Print pour positionner la sortie.

Syntaxe de l'instruction MsgBox("invite" , **code-bouton** , "titre")

Avec un seul chiffre, on peut faire 4 choix d'un coup : on additionne simplement les 4 chiffres choisis.

Le premier groupe de valeurs (0 à 5) décrit le nombre et le type de boutons affichés dans la boîte de dialogue.

Le deuxième groupe (16, 32, 48 et 64) décrit le style d'icône accompagnant le message.

Le troisième groupe (0, 256 et 512) définit le bouton par défaut.

Enfin, le quatrième groupe (0 et 4096) détermine la modalité de la zone de message.

Au moment d'ajouter ces nombres pour obtenir la valeur finale de l'argument code-boutons, ne sélectionnez qu'un seul nombre dans chaque groupe.





Le problème de cette méthode est qu'il est **difficile de relire** son programme :

si je vois 257, il est assez malaisé de deviner que ce chiffre correspond à 256 + 1... C'est encore plus difficile **si je lis 309 : il faudrait reconstituer 256 + 48 + 5...** illisible et irritant.

Voilà pourquoi **on décrit la somme de constantes** au *nom clair*, plutôt que le simple résultat de l'addition : la relecture des choix est alors instantanée :

au lieu d'écrire 309, on écrira
vbRetryCancel + vbDefaultButton2 + vbExclamation.

Codes des boutons

Val.	Constante	Bouton choisi
0	vbOKOnly	Affiche le bouton OK uniquement.
1	VbOKCancel	Affiche les boutons OK et Annuler .
2	vbAbortRetryIgnore	... boutons Abandonner , Répéter et Ignorer .
3	VbYesNoCancel	Affiche les boutons Oui , Non et Annuler .
4	VbYesNo	Affiche les boutons Oui et Non .
5	VbRetryCancel	Affiche les boutons Répéter et Annuler .
0		N'affiche aucune icône particulière
16	VbCritical	Affiche l'icône Message critique : 
32	VbQuestion	Affiche l'icône de confirmation : 
48	VbExclamation	Affiche l'icône Message d'avertissement : 
64	VbInformation	Affiche l'icône Message d'information : 
0	VbDefaultButton1	Le premier bouton est le bouton par défaut .
256	VbDefaultButton2	Le deuxième bouton est le bouton par défaut .
512	VbDefaultButton3	Le troisième bouton est le bouton par défaut .
0	VbApplicationModal	Application modale. L'utilisateur doit répondre au message affiché dans la zone de message avant de pouvoir continuer à travailler dans l'application en cours. Alt Tab est autorisé pour zapper vers un autre programme.
4096	vbSystemModal	Système modal. Toutes les applications sont interrompues jusqu'à ce que l'utilisateur réponde au message affiché dans la zone de message (Alt Tab interdit et multitâche suspendu)