DEBUTER AVEC VISUAL BASIC POUR APPLICATIONS (VBA5)

Auteur: Christian Herbé
Date: Décembre 1999
MAJ: Juin 2001

Le présent document est un support de cours. L'objectif de la formation est d'aider les stagiaires à prendre en main ce langage. Puisque l'utilisateur a la possibilité d'enregistrer ses actions, on se limitera ici à l'étude du code permettant le contrôle et l'intervention dans le déroulement des programmes.

Un peu au delà de l'iniation:

Pourquoi le terme de macro ?

Macro sous entends macro commandes c'est à dire un ensemble de commandes destinées à réaliser une ou plusieurs actions sans intervention de l'utilisateur. Les macros sont aux logiciels ce que les scripts ou "batchs" sont aux systèmes d'exploitation.
Très souvent, dès qu'un logiciel comporte un nombre important de commandes ou de fonctions distinctes, il est accompagné d'un langage de macro programmation . MS Office ne déroge pas à cet usage. La particularité d'une macro est d'être "attachée" à un logiciel spécifique. Il est impossible de l'exécuter depuis un autre logiciel que celui qui l'a créée (il y a cependant une exception partielle avec VBA5).

Historique du langage.

Le 1er tableur de Microsoft, Multiplan, était déjà doté d'un langage de programmation. Son successeur, Excel mais également les autres applications de bureautique ont bénéficié d'un tel outil. Jusqu'à la version 4 d'Excel, la macro programmation consistait à écrire une suite de fonctions dans une feuille spécifique. Les fonctions ou mots clés des

macros avaient la même syntaxe que les fonctions du tableur. Le code ainsi écrit était exécuté séquentiellement c'est à dire ligne par ligne et n'était en aucun cas transposable à une autre application Microsoft. Dans le même temps, MS proposait des langages de proprammation destinés à écrire des programmes autonomes ou directement exécutables depuis un système d'exploitation (on parle alors de langages compilés). Ces langages étaient aussi de type séquentiels jusqu'à QuickBasic inclu. Les langages séquentiels étaient assez bien adaptés aux systèmes dit de "commandes en lignes" comme MS-DOS mais il était difficile de développer des applications graphiques.
Au début des années 90, la firme de Seattle a donc proposé un nouveau langage pour le développent d'applications autonomes: Visual Basic (VB). Il est conçu pour programmer dans un environnement graphique comme Windows.
A partir de la version Excel5, MS à remplacé le langage de macro programmation propre à chacune de ses applications par des adaptations de Visual Basic à MS-Office rédigé en "langue étrangères" . Autrement dit, il existait un langage Visual Basic Applications pour chacune des langues dans lesquelles MS-Office était écrit. Cela a contraint Microsoft à écrire un traducteur de macros afin de permettre aux programmes écrits dans une langue d'être exécuté dans une application d'une autre langue.
Malheureusement, ce traducteur fonctionna tellement mal que le fabriquant décida de supprimer l'adaptation aux "langues étrangères" c'est pourquoi VBA5 est écrit uniquement en Anglais.

Quelques mots sur la Programmation Orientée Objet (POO)

La définition d'un objet en programmation est différente de celle des objets de la vie quotidienne puisqu'il s'agit généralement d'objets virtuels. Mais comme dans la vie, un objet a des propriétés et des méthodes. Une voiture a des propriétés: la couleur, la carrosserie, le moteur. Elle a des méthodes: accélerer, freiner, rouler ...
Il en va de même pour un objet informatique. Un document Excel a des propriétés:son nom, sa version... Il a des méthodes: ajouter une feuille.
On appliquera la méthode "add" à l'objet workbook (document) et l'on déterminera ses propriétés en lui attribuant un nom, des options de protections ...
Créer un objet en lui attribuant des propriétés définies s'appelle l'instanciation.
L'intérêt de la POO est qu'il n'est pas necéssaire de connaitre toutes les propriétés d'un objet pour l'utiliser mais seulement celles sur les quelles on veut agir.
Un objet peut contenir d'autres objets. C'est le cas de l'objet application (Excel par exemple) qui peut contenir les objets worbooks qui eux mêmes peuvent contenir des objets worksheets (feuilles) qui eux mêmes contiennent des objets ranges (cellules ou plages de cellules). On parle alors de conteneurs.

Comment situer VBA parmi les autres langages ?

Il n'est évidement pas possible de citer tous les langages de programmation. On ne parlera que des outils les plus utilisés à l'INRA.

Classement par type de langage:

Séquentiel	**Structuré**	**Intermédiaire**	**POO**
scripts, shell-scripts et batchs	C	VB	Java
macros XL et Word	Pascal	VBA	C++
SAS , S+			VB *
basic			VBA*
Fortran standard			

** L'appartenance de Visual Basic au groupe POO est quelques fois l'objet de polémiques.*

Un programme compilé est du code directement lisible par le système . "On parle alors de langage machine". La phase de compilation consiste à traduire du code "intelligible" à l'homme en code binaire. L'élaboration d'un programme compilé comprends donc une étape supplémentaire par rapport aux langages interprétés mais son exécution est plus rapide. En dehors de Java et du C, tous les langages compilés cités ici sont des produits commerciaux.
Le code d'un langage interprété est exécuté (interprété) en l'état ou il a été écrit mais son déroulement étant dépendant d'un ou plusieurs logiciels, il est sensiblement plus lent.
Les produits dit "interprétés" sont gratuits ou fourni avec leur logiciel "maitre".

Classement par type de fonctionnement:

interprétés interprétés compilés

Dépendent uniquement d'une famille d'OS:	**Dépendent de l'OS ET d'un logiciel:**	**dépendent généralement d'une famille d'OS**
Scripts, Shell-scripts et Batchs	macros	Java (ne dépend pas de l'OS mais d'une machine virtuelle)

PERL	VBA	C , C++
TCL/TK (existe maintenant sur Mac, Win32 et Unix)		Fortran , Pascal
		Basic , Visual Basic

Comment choisir un langage de programmation ?

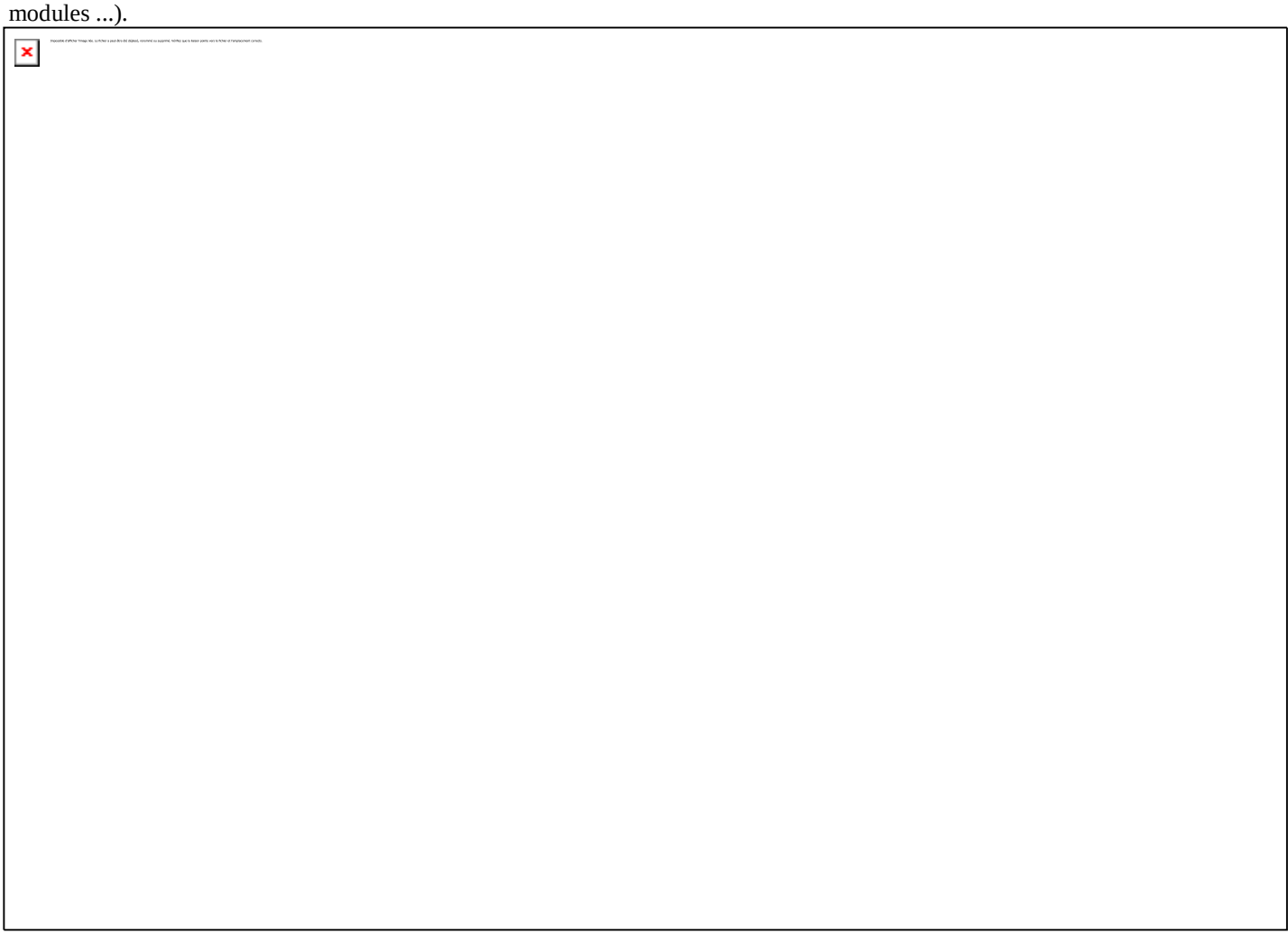
On devrait plutôt se demander: comment puis-je me positionner par rapport à tous ces langages tant la question est vaste. Aussi, je me bornerai à donner quelques conseils.

Objectif	motivation	langage correspondant	somme des gradients de difficulté
Suivre ou développer ou des applications "lourdes et performantes"	dictée par la profession	C++	200
Applications lourdes et portables sur diverses plateformes	dictée par la profession	Java	200
Applications de toutes natures	Enrichir son CV	C++ et/ou Java, VB	180
Applications durables, moyennement compliquées		C++ ou Java ou VB	150
petites applications performantes et agréables, demandant un temps d'apprentissage raisonable		VB, TCL/TK	100
Applications très spécialisées (Stats, BDD ...)	répondre à un besoin spécifique	VB, VBA, macros, scripts, SAS ou S+	100
Automatiser des tâches repétitives, faciliter l'usage d'un logiciel	répondre/anticiper des besoins particuliers	VBA, macros, scripts	60

Langages	Avantages	Inconvénients
C++	Un des plus performants, portable moyennant qlq aménagements	ardu, assez cher
Java	Gratuit, très répandu, portable sur Unix, Win32 et Mac. Ne présente pas de difficultés pour les personnes qui connaissent déjà C, C++	parfois lent pour des applications lourdes, le code pour le graphisme est complexe
Visual Basic	Relativement facile à apprendre. Prix abordable	Reservé exclusivement aux PC, sa toute relative simplicité peut entrainer des erreurs graves et compliquer le développement
TCL/TK, PERL, shell-scripts	gratuits, rapidité d'apprentissage pour de petites applis et performants, surtout le graphisme sous TCL/TK	très dépendants d'un OS y compris dans la syntaxe donc difficiles à distribuer
VBA	Pas d'achat supplémentaire, la solution la plus facile à apprendre. Un excellent tremplin vers un langage plus difficile.	dépend de MS-Office dans sa version actuelle donc ne tourne pas dans les versions antérieures. Comme tous les langages MS, son avenir est incertain
SAS, S+	Ce qui se fait de mieux dans le traitement de données. Apprentissage accessible à tous. Très répandus dans le domaine scientifique mondial et les grandes entreprises	2 logiciels chers. Sans intérêt en dehors du traitement de données.Leur langage respectif commence à vieillir.

Prise en main de l'éditeur VBA5

On accéde à l'éditeur depuis les menus d'application MS-Office, XL pour ce qui nous concerne:
menbu Outils puis *Macros* puis *Visual Basic Editor* ou en appuyant simultanément sur ALT+F11.
L'écran est divisé en cadres.L'explorateur de projets liste les documents ouverts et leur compositions (feuilles, modules ...).



La fenêtre de propriétés indique les propriétés du document sélectionné. Si la fenêtre de code n'est pas à l'écran, double-cliquez sur *ThisWorkbook* dans l'explorateur d'objets. Nous verrons plus tard comment appeler d'autres eléments de l'éditeur VBA5. L'explorateur d'objet a 3 boutons qui permettent de se déplacer dans les objets.
L'éditeur est semblable à un petit logiciel de traitement de texte avec des fonctions propres. Si vous écrivez dans le module ***sub nom()***, il ajoute automatiquement 2 lignes plus bas les mots clés ***End Sub*** et place le curseur à une tabulation en retrait. Il place également des commentaires en cas d'enregistrement. Tout ce qui est placé à gauche d'une apostrophe (') est un commentaire. L'éditeur les écrit en vert.Vous pouvez bien entendu personnaliser ces paramètres de couleur.
Le cacactère invisible de fin de ligne est traité comme la fin de l'instruction. Lorsqu'une instruction ne peut pas tenir sur une seule ligne vous indiquez au système de lire la suite dans la ligne suivante en mettant un espace et un "tiret bas" (_) après le dernier caractère de la 1ère ligne.
Mise en forme du code.
Bien que tout à fait facultatif, il est d'usage de mettre en retrait les instructions faisant partie d'un bloc. Cela améliore la lisibilité du texte.

Premières macros

Enregistrement de la 1ère macro
Revenez dans la feuille XL.
Il est très pratique de pouvoir changer le mode de référencement des cellules en cours d'enregistrement. Assurez vous que la barre d'outils "*Arrêter l'enregistrement de macro*" est cochée depuis le menu *Affichage* puis *barres d'outils* puis *Personnaliser* et enfin l'onglet *Barres d'outils*. Tapez une valeur quelconque dans une cellule. Faites en sorte que la cellule soit sélectionnée. Allez dans le menu *Outils* puis *Macro* et *Nouvelle macro*. Dans le champ *Nom de la macro*, écrivez essai1. Nous verrons plus loin comment utiliser les autres champs. Cliquez sur *Ok* .

Allez dans le menu *Format* puis *cellule* cliquez sur l'onglet *Police* choisissez gras.
Allez dans le menu *Outils* puis *Macro* et choisissez *Arrêter l'enregistrement.*
(En fonction de votre configuration, il est possible que la *barre d'outils macros* s'affiche à l'écran. Dans ce cas, utilisez ses boutons)
Activez l'éditeur VBA. Vous constatez que le document actif contient un élément supplémentaire nommé *Modules* qui lui même contient *Module1*. Double-cliquez le. Vous voyez le code que vous venez d'enregistrer.
Refaite un enregistrement nommé "essai2" en utilisant non pas le menu Format ... Mais en cliquant sur l'icône "*Gras*" et comparez le code de essai1 et essai2. Vous voyez que la macro "essai2" est beaucoup plus compacte!
Dans essai1, VBA a renseigné chaque option (les propriétés de l'objet) du menu Format alors que essai2 s'est limité aux instructions de mise en forme du texte. Tenez compte de cette différence avant d'enregister une macro. Vous obtiendrez un code beaucoup plus facile à comprendre!
Testez maintenant ces deux programmes:
Dans la feuille XL, entrez une valeur dans une autre cellule et allez dans le menu *Outils* puis *Macro* et encore *Macros*. Sélectionnez essai1 et cliquez sur *Exécuter*. recommencez ensuite avec le programme essai2.

Ecrire la 1ère macro

Généralement, la 1ère leçon de programmation commence par l'écriture du programme: "Hello World". Je n'ai aucune raison de vous éviter cela!
Allez dans l'éditeur VBA. A la fin de "essai2", écrivez:

```
Sub hello()
MsgBox "Salut tout le monde", vbOKOnly, "hi"
End sub
```

Revenez dans Xl et exécutez la macro hello.
Depuis l'éditeur, copier le l'instruction: MsgBox "Salut tout le monde", vbOKOnly, "hi" dans essai2 et exécutez cette dernière.
Vous venez de voir 2 aspects complémentaires de VBA. La macro essai2 contient maintenant du code dont une partie a été enregistré et le reste a été écrit par vous même. Développer en VBA consiste donc à enregistrer tout ce qu'il est possible et à compléter par du code écrit "à la main". Le développeur devra écrire le code quand il voudra:
- Afficher une boite de dialogue pour informer l'utilisateur ou lui demander d'intervenir
- Créer des variables
- mettre des tests dans le programme.

Ce sont ces 3 points que nous allons détailler mais auparavant, regardons la structure de nos 3 programmes.
Vous avez pu constater que chaque programme commence par *sub()* et se termine par *End sub* . *Sub* doit être interprété comme sous routine. Quand vous écrivez le mot clé *sub*(), l'éditeur l'interpréte comme l'annonce d'un nouveau programme et il écrit automatiquement *End sub* sur la ligne inférieure. Sub doit être suivi par le nom de la macro. Ce nom doit commencer par une lettre.

La commande msgbox

Sélectionnez le mot clé msgbox dans l'éditeur VBA et appuyez sur la touche F1 pour obtenir de l'aide. Voici la syntaxe:
MsgBox(prompt[, buttons] [, title] [, helpfile, context])
prompt est le message contenu dans la boîte de dialogue.Le texte doit être écrit entre guillemets Anglais (" ").
Les crochets signalent que cet argument est facultatif.
buttons vous permet de choisir un type de bouton. *VbOkOnly* affiche une seul bouton. *VbOkCancel* affiche 2boutons. Chacun de ces boutons renvoie une valeur que vous pourrez exploiter par exemple avec un test *IF* (si) pour exécuter une action en fonction de la réponse de l'utilisateur. Vous pouvez voir dans l'aide qu'il y a beaucoup de types de boutons disponibles.
title donne un titre à la boîte qui doit être écrit lui aussi entre guillemets
Les arguments de commandes doivent toujours être séparés par des virgules.
Pour en savoir plus, voir le chapitre: Les boîtes de dialogue

---

Enregistrement et sauvegarde des programmes

Depuis XL, allez dans les menus Outils puis Macros et nouvelle macro.
Le champ *Touche de raccourci* vous permet d'affecter un raccourci clavier à la macro que vous vous apprêter à enregistrer. Afin de limiter les risques de création de raccourcis déjà affectés dans le tableur, il est conseillé d'utiliser pour les macros des raccourcis utilisant les touches Ctrl+Shift+lettre.
Le champ *Enregistrer la macro dans*: permet de choisir la destination du programme. Si elle est enregistrée dans "c*e classeur*", elle ne sera disponible que si le fichier est ouvert. Si elle est enregistrée dans le classeur de macros personnelles, elle sera disponible chaque fois que XL sera lancé. Il y a cependant une différence en fonction des

systèmes d'exploitation "sécurisés" ou non. Sous NT et dans certains cas sous W9*, il existe un classeur de macros personnelles pour chaque utilisateur.

Intervention de l'auteur dans le code des programmes

VBA5 étant un langage macro ayant la faculté d'enregistrement des programmes, le rôle de l'auteur est très limité par rapport aux autres langages. Cependant il est loin d'être insignifiant. Ce paragraphe est destiné à rappeler quels types d'interventions doivent être réalisées pour rendre les programmes fonctionnels.
Se positionner dans le document (Les adresses de cellules ou de plages):
Pendant l'enregistrement, les déplacements sont notés en position relative de la cellule active ou en références absolues par rapport au coin supérieur droit de la feuille mais il est très fréquent d'avoir à modifier ces valeurs.
Mémoriser une valeur afin de la réutiliser (les variables):
Exécuter une action en fonction d'une condition, d'un état, d'une valeur (les conditions):
Vous aurez souvent besoin d'exécuter un programme en fonction d'une valeur, par exemple traiter différement les lignes d'une feuille en fonction d'une valeur précise (nom d'élèves, numéros de placettes Etc). Dans ce cas, vous devrez utiliser les conditions ***IF*** ou ***Select Case***.
Paramétres de l'utilisateur (les boîtes de dialogue):
Une macro n'est réutilisable que si l'utilisateur peut entrer des valeurs ou des références de cellules ou des noms de fichiers. Les boîte de dialogues et les userforms sont des moyens de communication avec l'utilisateur (les userforms ou feuilles VBA destinées à la création d'interface graphiques complexes ne sont pas traitées dans ce document en raison de leur complexité pour des débutants) .
Une partie de programme doit être répétés (Les boucles):
L'utilisateur peut fournir des informations inadaptées ou exécuter le programme en dehors de son champ (Gestion des erreurs)

Les variables

Les variables sont destinées à stocker des valeurs en vue de leurs réutilisations ultérieures. Elles sont de plusieurs types. La méthode de déclaration est la suivante:
dim nom_de_variable *As* type ; exemple de variables contenant du texte:
dim* toto *As string crée la variable toto et la déclare de type chaîne de caractères mais toto est vide. Vous pouvez maintenant lui affecter une valeur:
toto = "Hello World"
la variable toto existe et a pour valeur "Hello World". Vous pourriez écrire le programme hello comme ceci:

```
sub hello2()
dim txt as string ' déclaration
dim titre as string ' déclaration
txt = "Hello World" 'affectation
titre="Salut" 'affectation
MsgBox txt, vbOKOnly, titre 'utilisation des variables dans une instruction
end sub
```

L'avantage de ce programme est de vous permettre de réutiliser ultérieurement les variables sans être obligé de reécrire le contenu et votre code gagne en "lisibilité".
Tout ce qui est à droite d'une apostrophe (') est du commentaire et n'est pas lu par le programme. Vous n'êtes pas obligés de déclarer la variable. Vous pouvez directement lui affecter une valeur ex:

```
sub hello3()
txt = "Hello World"
titre="Salut"
MsgBox txt, vbOKOnly, titre
end sub
```

Quand vous procédez ainsi, VBA déclare la variable pour vous. Vous pouvez aussi méler du texte et une variable dans une instruction, ex:

```
sub hello4()
dim txt as string
dim titre as string
txt = "Hello World"
```

titre="Salut"
MsgBox "Exemple de texte et de variable: "& txt, vbOKOnly, titre
' le symbol et commercial (&) lie la chaîne et la variable
end sub
Chaque fois que vous déclarez une variable, le système lui réserve un espace mémoire. Comme pour les variables "string", vous n'êtes pas obligé de les déclarer ni de leur affecter une dimension. Cependant, si vous devez développer des programmes gourmands en ressources ou si vous voulez les exécuter sur des systèmes limités, vous avez intérêt à définir précisement vos variables. Pour limiter la longueur d'une chaîne:
dim variable as string** * **nb_caractères ex: ***dim txt as string * 11*** lime la chaîne à 11 caractères.
Les variables numériques:

Type	valeurs acceptées	mémoire occupée
Byte	entier de 0 à 255	1 octet
Integer	entier entre -32768 et 32767	2 octets
Long (entier long)	entre -2 147 483 648 et 2 147 483 647	4 octets
Single	nombre à virgule flottante	4 octets
double	double précision	8 octets
Currency	à virgule fixe avec 15 chiffres pour la partie entière et 4 pour la partie décimale	8 octets

Les variables de type matrice ou de type array encore appelées variable tableau
Les variables que nous avons vu jusquà présent ne stockent qu'une seule valeur. Les matrices stockent une liste d'éléments. Il est obligatoire de les déclarer:
dim nom(nb_ éléments) as type
ce qui revient à :
dim toto(5) as integer déclare la variable tableau de type "entier" ayant la possibilité de recevoir 5 valeurs. Attention, l'index commence à zéro. Avec ce type de déclarations, les éléments du tableau seront indexés de 0 à 4. Vous pouvez modifier l'index dans la déclaration:
dim toto(1 to 5) as integer
Exemple de création de matrice:
sub matrice1()
dim devinette(1 to 8) as string
devinette(1) = "M et Mme Froid ont 7 enfants, quels sont leurs prénoms ?"
devinette(2) = "Eva"
devinette(3) = "Aude"
devinette(4) = "Dan"
devinette(5) = "Marc"
devinette(6) = "Samson"
devinette(7) = "Gilles"
devinette(8) = "Ella"
'*fin de la création du tableau*
' *Ce qui suit est une boucle qui va afficher8 fois une boite de dialogue avec chacune des valeurs du tableau*
for compteur = 1 to 8
msgbox devinette(compteur)
next compteur
end sub
L'exemple précédent utilise une matrice unidimensionnelle mais un tableau peut être multidimensionnel exemple:
dim tab(1 to 10, 1 to 10) as integer
crée un tableau à 2 dimensions où chaque vecteur contient 10 valeurs. Ce type de tableau est très utilisé pour stocker des valeurs lues dans une plage de feuille XL. Vous pourrez créer des tableaux ayant autant de lignes et de colonnes que la plage.
Accéder aux valeurs d'un tableau
Si une seule valeur vous intéresses et que vous connaissez ses coordonnées dans le tableau, il suffit d'écrire:
A=tableau(3) pour affecter la variable "A" de la 3ème valeur de tableau(). Nous verrons dans le chapitre sur les boucles comment rechercher une valeur dans un tableau.
Les variables de type object
Un objet peut désigner un classeur (workbook), une feuille (sheet) ou tout autre partie de l'application. Une fois la variable de type object declarée (obligatoire dans ce cas), vous lui affecter une valeur avec le mot clé *SET* :
set C = activeworkbook affecte la variable C du nom du document actif

Portée et durée des variables

Jusqu'à présent, nous avons déclaré les variables entre les mots clés *sub*() et *end sub*. Cette pratique limite la portée des variables à ce module. Si vous appelez plusieurs modules depuis un projet et que vous souhaitez rendre certaines variables utilisables par tous les modules, vous devrez soit les faire précéder du mot clé *public* , soit les déclarer à l'extérieur d'un module.

Références des cellules

Se déplacer dans la feuille est interprété comme de la manipulation de l'objet ***range*** qui représente une ou plusieurs cellules. En plus de range, vous utiliserez ***cells*** qui est une collection de toutes les cellules du document actif. Les propriétés ***row et column*** donnent le numéro de la ligne et de colonne. ***Activecell*** renvoie un objet range indiquant la cellule sélectionnée. ***select*** sélectionne un objet. ***offset*** décale par rapport à une plage indiquée et ***activate*** active un objet .

Références absolues

C'est le mode d'enregistrement par défaut. Le bouton *Enregistrement relatif* de la barre d'outils macro permet le passage d'un mode à l'autre.

L'expression: **range("A1").select** sélectionne ou active la cellule A1.

L'expression: **range("A1:B10").select** sélectionne une plage de A1 à B10

L'expression: **range("A1, B10").select** sélectionne les cellules non contiguës A1 et B10.

Le code:

range("A1, B10").select
range("B10").activate

sélectionne les cellules A1 et B10 et donne le focus à la cellule B10.

Ce type de références est autorisé même si la feuille active a un système de notation de type L1C1

Pour faire référence à une seule cellule, vous pouvez aussi utiliser:

cells(1,1).select 'sélectionne la cellule A1

Attention cependant, les valeurs entre les parenthèses (n° de ligne et n° colonne) sont des références absolues. cells(1,1) indique la ligne1 et colonne 1. Vous pourriez aussi remplacer ces valeurs par des variables.

Références relatives

Ce type de références n'est pas très bien fait dans VBA5 et constitue la principale difficulté du débutant.

Une référence relative est généralement une adresse à partir de la cellule active. L'expression:

activecell.offset(0,1).range("A1").select

ou

activecell.offset(0,1).select

déplace (offset) la sélection vers une cellule placée sur la même ligne et une colonne à droite.

Pour se déplacer vers la gauche ou vers le haut, il suffit de donner des valeurs négatives:

activecell.offset(0,-1).select

pour sélectionner une plage de 4 cellules dans la colonne voisine de gauche:

ActiveCell.Offset(0, -1).Range("A1:A4").Select

Quelques exemples de sélection de cellules ou de plages:

Sélection de toutes les cellules non-vides et contiguës de la ligne 1:

Range("A1", Range("A1").End(xlToRight)).Select

Sélection de toutes les cellules non vides autour de la cellule "A1" :

Range("A1").CurrentRegion.Select

*(produit le même résultat que le raccourci clavier: Ctrl + *)*

Sélection de toutes les cellules non vides autour de la cellule active:

Range(ActiveCell, ActiveCell.CurrentRegion).Select

*(produit le même résultat que le raccourci clavier: Ctrl + *)*

Sélection de toutes les cellules non vides autour de la cellule active et exclusion de la ligne 1:

Dim Plage As Range
With Selection.CurrentRegion
Set Plage = .Offset(1).Resize(.Rows.Count - 1, .Columns.Count)
End With

(produit le même résultat que les raccourcis clavier: Ctrl + Shift + flèches)

Sélection de toutes les cellules non vides dans une seule colonne en dessous de la cellule active:

Range(ActiveCell, ActiveCell.End(xlDown)).Select

(produit le même résultat que le raccourci clavier: Ctrl + Shift + flèche en bas)

Sélection de toutes les cellules non vides sur une seule ligne et à droite de la cellule active:
Range(ActiveCell, ActiveCell.End(xlToRight)).Select
(produit le même résultat que le raccourci clavier: Ctrl + Shift + flèche en à droite)
(Utilisez les instructions **xl*ToLeft*** *et* **xlUp** pour les autres directions)
Séléction de la dernière cellule non vide de la colonne:
ActiveCell.End(xlDown).Select
Sélection de la dernière cellule du tableau:
Range("A1").SpecialCells(xlLastCell).select
(produit le même résultat que le raccourci clavier: Ctrl + touche fin)
Sélection de la dernière cellule contenant une valeur:
Cells.Find("*", [A1], , , xlByRows, xlPrevious).Select
Déplacer la sélection d'une colonne vers la droite:
Selection.Offset(, -1).Select

Structurer les projets

Avec VBA, on est très loin des listings de langages comme le basic ou le langage macro XL4. Le developpeur à tout intérêt à écrire des procédures courtes et à les appeler depuis un programme principale. Il est en effet possible d'utiliser la commande ***call*** pour appeler un module. Les parenthèses de l'instruction ***sub hello()*** sont là pour passer des paramétres à la procédure appellée. Vous pouvez structurer vos programmes de cette manière:

```
sub hello() ' Programme principal
call boite("Bonjour") ' fonctionne aussi avec: boite("bonjour")
end sub
sub boite(txt) ' programme auxiliaire
msgbox txt
end sub
```

Dans ce cas, le programme ***hello*** appelle le programme ***boîte***tout en lui passant un paramétre ("**bonjour**").
Dans les exemples precédents, nous avons reécrit l'instruction **msgbox** dans chaque programme.En écrivant de petites procédures spécialisées, on peut les appeler autant de fois que l'on veut sans les reécrire tout en leur faisant réaliser une action légèrement différentes à chaque fois. L'autre avantage au moins aussi important réside dans le fait qu'un projet "éclaté" en petits modules est beaucoup plus lisible donc la recherche d'erreur y est beaucoup plus facile. Dernier avantage et non des moindres, vous pouvez consacrer un peu de temps à peaufiner une procédure si c'est pour la réutiliser souvent.

Les différents types de procédures.

Je n'en citerai que deux.

Procédures sub

J'en ai déjà parlé au début. Ce sont des sous-routines qui exécute une série d'instructions mais ne renvoient pas de valeur en fin d'exécution. Elles sont structurées comme ceci:

```
sub nom_de_la procedure()
instructions
...
End sub
```

Le nom ne peut pas contenir plus de 255 caractères, il doit commencer par une lettre et ne pas contenir les 5 caractères suivants:
@ & $ # !
En outre, le nom doit être différent des mots clés du langage.

Procédures Function

Les fonctions contiennent elles aussi une série d'instructions et renvoient une valeur. Voici leurs structures:

```
function nom_fonction(variable ayant reçu une valeur de la proc appelante)
instructions
...
nom_fonction=expression
end function
```

Exemple:

```
function age_au_carre(age_capitaine)
age_au_carre=age_capitaine*age_capitaine
end function
```

Si vous appellez cette fonction depuis une procédure sub, vous devez lui passer une valeur: *call age_au_carre(10)* que la fonction utilise puis renvoie le résultat à la procédure appelante
Mais les fonctions ont aussi un autre usage. Depuis XL, allez dans le menu *Insertion* puis *Fonctions* puis *Fonctions personnalisées*. Normalement, vous devez voir la fonction *age_au_carre*. vous venez d'ajouter une fonction au

tableur. Si vous souhaitez en disposer à chaque session, copiez la dans le classeur de macros perso.xls.
Attention:
vous devez ajouter l'instruction **Application.Volatile** pour que la fonction se recalcule automatiquement lors d'un changement de valeur de l'une des cellules utilisées ce qui donne ceci:
function age_au_carre(age_capitaine)
Application.Volatile
age_au_carre=age_capitaine*age_capitaine
end function

Les boucles

"tant que ..."
Une boucle est un moyen de repéter une instruction ou un bloc d'instructions autant de fois que c'est necéssaire.
Do ... Loop et While...Wend sont généralement utilisées pour repéter une instruction sur tout le document.
For each ... Next est utile pour une action sur une collection.
While condition
instructions
Wend
Vérifie d'abord la condition et exécute les instructions si condition renvoie true (vrai)
Do ... Loop est plus souple parce que la condition peut figurer aussi bien avant les instructions qu'après.
Exemple:
le test est réalisé avant les instructions.
Sub deplacement()
Do while activecell.value <>"" ' tant que la cellule active n'est pas vide
selection.offset(1,0).select
loop
End sub
Le test est réalisé après les instructions.
Sub deplacement2()
Do
selection.offset(1,0).select
loop while activecell.value <>""
End sub
La boucle est réalisés tant que la cellule n'est pas vide.
Sub deplacement3()
Do While IsEmpty(ActiveCell) = False
Selection.Offset(1, 0).Select
Loop
End Sub

La boucle "pour ... suivant"

For ... Next repéte la même action un nombre de fois déterminé par l'utilisateur ou le programmeur.
La syntaxe est: ***For*** compteur = début ***to*** fin ***step next*** compteur
Compteur est une variable à créer. Début est la valeur initial de compteur. fin est la dernière valeur de compteur.
step est le pas de la boucle si différent de un. Exemple:
Sub deplacement4()
For compteur = 1 to 5 step 2
Selection.Offset(1, 0).Select
Next compteur
End Sub
Sélectionne 3 fois la cellule inférieure.
Il est également possible d'imbriquer des boucles. C'est fort utile quand on doit travailler sur des tableaux à plus d'une dimension. Exemple: votre feuille XL contient dans une colonne des prénoms d'élèves et dans l'autres des notes. Vous souhaitez stocker ces valeurs dans une variable matrice à deux dimensions. Voici le tableau:

Jean	10
Oléane	18
Muriel	7
Luc	13

Sylvie	5

Le programme suivant lit les données dans la feuille et les stocke dans la variable matrice "tableau"

```
Sub stock_tableau()
Dim tableau(5, 2)
For lignes = 1 To 5
For colonnes = 1 To 2
tableau(lignes, colonnes) = Cells(lignes, colonnes)
Next colonnes
Next lignes
End Sub
```

Dans la réalité, ce programme a un intérêt limité puisqu'on connait rarement les dimensions de la feuille de données. L'exemple suivant compte le nombre de lignes et colonnes de la sélection pour dimensionner la variable matrice (la cellule active doit être A1):

```
Sub stock_tableau2()
Dim tableau()
nb_lignes = Range("A1").End(xlDown).Row
nb_colonnes = Range("A1").End(xlToRight).Column
ReDim tableau(nb_lignes, nb_colonnes)
For lignes = 1 To nb_lignes
For colonnes = 1 To nb_colonnes
tableau(lignes, colonnes) = Cells(lignes, colonnes)
MsgBox tableau(lignes, colonnes)
Next colonnes
Next lignes
End Sub
```

Cette syntaxe parait compliquée pour stocker des valeurs accessibles dans le document Excel. Mais que l'on ne s'y trompe pas, l'accès au données d'un tableau est infiniment plus rapide que la lecture dans une feuille et vous affranchit des problèmes de référence des cellules; d'autres parts, si vous devez un jour apprendre un langage compilé, cette pratique sera indispensable!

La boucle "pour chaque ... suivant"

La boucle **For Each** élément ...
instructions
Next
est utilisée pour un traitement sur une collection d'objets. Dans l'exemple précédent, tableau() était une collection de valeurs.

La procédure suivante lit chaque cellule d'une plage sélectionnée et met en gras les valeurs supérieures à 10.

```
Sub gras_maigre()
Dim cellule As Range
For Each cellule In Selection.Cells
If cellule.Value >= 10 Then
cellule.Font.Bold = True
else
cellule.Font.Bold =false
End If
Next cellule
End Sub
```

Nous venons de voir un des très gros avantages de VBA sur l'ancien langage XL. Avec l'ancienne version, il était impossible d'effectuer une action sur une cellule sans qu'elle soit sélectionnée ce qui obligeait le programme à se déplacer sans cesse et ralentissait l'exécution. N'hésitez pas à utiliser les variables tableaux et les boucles pour agir sur une collection plutôt que d'obliger le programme à activer chaque élément. En outre, vous ne serez pas perturbé si un jour vous deviez passer à un langage compilé.

Les conditions

Si ... Alors ...Sinon ...

Elle peut prendre plusieurs formes:
If condition **then**
instructions ..
End If

ou
If condition **then**
instructions ..
else
instructions...
End If
ou
If condition then
i**nstructions ..**
else
instructions...
elseif autre condition
instructions..
End If
Nous l'avons déjà rencontré dans les exemples précédents. Avec la boucle **For ... Next** , ce sont vraisemblablement les instructions les plus utilisées.
Une boucle **If** doit toujours être suivi sur la même ligne de l'instruction **Then** et se terminer par **End if**.
Si la condition renvoie true(vrai), les instructions suivant **Then**sont exécutées. Sinon, la boucle cherche l'instruction **Else**. Si elle est trouvée, les instructions suivantes sont exécutées.
Comme nous avons déjà rencontré à plusieurs reprises la boucle **If** dans sa plus simple expréssion, il est inutile de la détailler davantage cependant, les formes que peut prendre la condition mérite de s'y arrêter:
If isempty(activecell) then
If activecell.value = "" then
Ces 2 formes renvoient *true* si la cellule active est vide, sinon elles renvoient *False*(faux).
On peut aussi utiliser le mot clé And dans une condition:

If activecell.value* >10 *And* *activecell.value* <20 *then
Dans ce cas, la condition renvoie *true* seulement si les 2 conditions sont remplies (notez qu'il faut repéter le nom de l'élément à tester: activecell).

If activecell.value <10 or activecell.value >20 then
La condition renvoie true si au moins une des 2 conditions est remplies.
Les blocs d'instrutions **If then** ou **If then else** ne vérifient qu'une seule condition alors que:
if condition1 then
instructions
elseif condition2
End If
Vérifie deux conditions. Vous pouvez emboîter ainsi autant de conditions que vous le souhaitez.

La structure Select Case

Elle s'avère très utile quand une condition peut renvoyer à plusieurs situations.
Select case expression
case 1
instructions
case 2
instructions
case n
instructions
case Else
instructions
End Select
La procédure suivante teste la valeur de la cellule active censée contenir un code postal et affiche le nom du département correspondant:
sub code_postal()
Select Case ActiveCell.Value
Case 54000 To 54999 'teste si la cellule contient une valeur entre 54000 et 54999 inclus
MsgBox "le département de Meurthe et Moselle "
Case 55000 To 55999
MsgBox "département de la Meuse"
Case 57000 To 57999
MsgBox "département de la Moselle"
Case 88000 To 88999

```
MsgBox "département des Vosges"
Case Else
MsgBox "la valeur ne corresponds pas à un département Lorrain"
End Select
End Sub
```
La valeur suivant l'instruction **Case** peut être de n'importe quel type. Pensez aux valeurs avec décimales. 1 est différent de 1,000001 et le comportement de **Select Case** sera différent.

L'instruction GoTo

Si vous voulez rendre vos programmes indeboguables, multipliez les instructions GoTo!
Au de là de 3 instructions, vous aurez les pires difficultés à vous y retrouver.
L'instruction GoTo est obligatoirement suivi d'une "étiquette" ou "balise". Une étiquette est le point de chute de GoTo. Si vous souhaitez que la procédure "saute" une ou plusieurs instructions, mettez une étiquette à l'endroit où vous souhaitez qu'elle se rende, exemple:

```
sub code_postal2()
question= msgbox("souhaitez vous continuer l'exécution de ce programme ?",vbOKCancel)
if question = 2 then GoTo fin ' cas particulier de IF: pas d'instruction End If en présence deGoTo
Select Case ActiveCell.Value
Case 54000 To 54999
MsgBox "le département de Meurthe et Moselle "
Case 55000 To 55999
MsgBox "département de la Meuse"
Case 57000 To 57999
MsgBox "département de la Moselle"
Case 88000 To 88999
MsgBox "département des Vosges"
Case Else
MsgBox "la valeur ne corresponds pas à un département Lorrain"
End Select
fin:
End Sub
```

Si l'utilisateur clique sur Annuler, "question" prend la valeur 2. Dans ce cas, **GoTo** exécute l'instruction suivant l'étiquette *fin:*
Notez que **GoTo** n'a aucun intérêt dans ce cas puisqu'il était plus simple d'écrire:

```
if question = 2 then exit sub
```

Malgrès tout le mal que j'ai pu en dire, **GoTo** reste incontournable dans un processus de gestion des erreurs. Nous le verrons plus loin.

Les boîtes de dialogue

Il faut faire la distinction entre les boîtes issues de fonctions VBA telles que **msgbox** et **inputbox** d'une part et les boîtes issues des méthodes de l'application d'autres parts. Nous avons suffisament utilisé la fonction **msgbox** pour ne pas revenir dessus.
La fonction **inputbox** permet de passer des valeurs de l'utilisateur au programme. En revanche, elle ne permet pas de spécifier un type de données ou de sélectionner des données. Puisque la méthode **inputbox** le permet, nous n'utiliserons que celle ci.
Inputbox est une méthode de l'objet application. Voici la syntaxe:

```
Application.inputbox(prompt, title, default, left, top, helpfile, helpcontextID,type)
```

Voyez l'aide pour connaitre les valeurs admises par l'argument type.

```
sub test_input()
set entree= application.inputbox(prompt:="Sélectionnez une cellule", _ ' Texte affiché dans la boîte
title:="Essai ", _ ' Titre
left:=3, _ ' Position horizontale
top:=-80, _ ' Position verticale
type:=8) ' Type d'entrée attendue
end sub
```

Vous pouvez également utiliser les boîtes de dialogues de l'application:

```
application.dialogs(XLdialogOpen).Show ' pour l'ouverture de fichier.
application.dialogs(XLdialogSaveAs).show ' pour enregistrer sous
```

Les opérateurs logiques

Opérateur description
Or renvoie vrai si l'une ou l'autre condition est vérifiée
And renvoie vrai si toutes les conditions sont vérifiées
Not négation
Il est possible d'utiliser plusieurs opérateurs dans une même expression à condition de les "enfermer" dans les parenthèses.:

If (condition1 Or condition2) And (condition3) then ...
Le test renverra *true* si une des deux premières conditions est vérifié ainsi que le 3ème.

Gestion des erreurs

Elles sont prévisibles si vos projets comportent des boîtes de dialogue. Si vous indiquez un type de données et que l'utilisateur en saisi un autre ou si un déplacement de la sélection devient impossible. Supposons que la cellule active soit A1.
si votre code contient une commande demandant au programme de sélectionner la cellule contiguë de gauche, cela génére une erreur. La procédure suivante est écrite pour déplacer la sélection vers la gauche:

```
Sub decale_gauche()
Selection.Offset(0, -1).Resize(Selection.Rows.Count, Selection.Columns.Count).Select
End Sub
```

Si cette macro est exécutée depuis la colonne "A", une erreur est signalée par VBA. Le code suivant prends en compte cet cas:

```
Sub decale_gauche2()
On Error GoTo ouste
Selection.Offset(0, -1).Resize(Selection.Rows.Count, Selection.Columns.Count).Select
Exit Sub
ouste:
msg = MsgBox(prompt:="déplacement impossible!", Buttons:=vbOKOnly, Title:="Erreur")
End Sub
```

Il faut placer la commande: **On Error GoTo** "balise" avant la commande susceptible de générer l'erreur.
Notez la présence de la commande **Exit sub**. Elle est indispensable pour sortir du programme si tout se déroule normalement.