

Informatique

Macros en VBA / Excel

Licence 3 TCI et Tourisme

A. Lemay

1 Introduction

Visual Basic for Applications (VBA) est le langage de programmation des applications de Microsoft Office. VBA permet non seulement d'automatiser les tâches que vous serez amenés à réaliser sous Excel - et ainsi vous faire gagner beaucoup de temps - mais également de réaliser de nouvelles tâches telles que :

- créer des applications complètes, plus claires et plus faciles à utiliser
- sécuriser vos saisies
- créer de nouvelles fonctionnalités à Excel

Le langage VBA est sensiblement le même que celui de Microsoft Visual Basic (VB). La différence étant que VBA dispose en plus de tout un ensemble de fonctionnalités propre à l'application concernée. Dans le cas de VBA pour Excel, il s'agit de tout un ensemble de fonctions permettant de manipuler cellules, feuilles de calcul. Il existe également de la même manière un VBA pour Word, Access et d'autres applications Microsoft.

Le langage VBA est accessible à tous. Cependant, une bonne connaissance d'Excel est nécessaire avant de se lancer dans la création d'application. En effet, il est important de bien maîtriser les principaux objets que manipule VBA. Depuis Excel 97, une application VBA est développée en Anglais. Ce ne doit pas être un frein pour ceux qui veulent débiter puisque peu de mots, rapidement familiers, sont nécessaires.

VBA, langage puissant, souple et facile à utiliser permet de réaliser très rapidement des applications qui vous feront économiser du temps et de l'argent.

2 Les débuts

Excel était initialement pourvu d'un système d'automatisation des tâches beaucoup plus sommaire que VBA. Il s'agissait de macro-commandes (ou macros), c'est-à-dire tout simplement de commandes Excel regroupant un ensemble de commandes de bases. Le terme est resté pour VBA et désigne une procédure qui effectuera un ensemble de tâches automatiquement.

Nous allons voir ici, tout d'abord, comment utiliser le système de macro d'Excel comme un simple enregistreur : vous effectuez une tâche que vous désirez automatiser une fois en l'"enregistreur" pour pouvoir ensuite ré-exécuter simplement la tâche autant de fois que vous le désirez. Nous verrons ensuite comment

2.1 Enregistrement de macro

Excel est doté d'un outil permettant de faciliter l'écriture des macros : l'enregistreur de macros. Cet outil permet de mémoriser une série d'opération effectuée par l'utilisateur pour les enregistrer dans le langage VBA. La macro ainsi enregistrée peut ensuite être exécutée (telle quelle ou après modification) pour reproduire la série d'opérations.

Pour enregistrer une macro, il suffit d'aller dans le menu **Outils** puis **Macros**. Sélectionnez **Nouvelle Macro**. Ensuite, à partir du moment où vous cliquerez sur Ok, toutes les actions que vous effectuerez, et uniquement celles-là, seront enregistrées et constitueront votre macro. La macro cessera d'être enregistrée lorsque vous cliquerez sur le bouton d'arrêt de l'enregistrement.

Exemples de macros :

- **Mise en forme de la zone sélectionnée** : après avoir cliqué sur ok, appuyez sur les boutons de mise en forme souhaitées (gras, couleur...) puis cliquez sur 'STOP'. Attention, ne cliquez à aucun moment sur votre feuille de calcul pour re-sélectionner une zone au risque que cette sélection soit enregistrée : votre mise en forme se fera alors toujours sur cette même cellule, puisque le fait de cliquer sur la cellule se reproduira dans la macro.
- **Changer de feuille** : pour faire une macro qui fait passer d'une feuille à une autre, rien de plus simple. Si vous voulez une macro qui vous emmène sur la feuille 1 par exemple, cliquez simplement sur l'onglet 'feuille 1' (comme vous le feriez habituellement pour aller sur cette feuille) au moment où votre macro est enregistrée.
- **Fonction de recherche** : vous pouvez de la même manière faire une macro qui reproduit n'importe quelle fonction d'Excel. Par exemple, une macro recherchant un texte 'truc' dans le document excel se fait en utilisant la fonction recherche (du menu éditer) au moment où votre macro est enregistrée.

En pratique

Pour enregistrer une macro :

- Allez dans le menu **Outils** puis **Macro** puis **Nouvelle Macro**
- Choisissez un nom pour votre macro
- Tapez sur 'Ok'. A partir de ce moment, toutes les actions que vous effectuez sont enregistrées dans votre macro
- une fois terminé, appuyez sur le bouton stop. si ce bouton n'apparaît pas, allez chercher la barre d'outils 'arrêter l'enregistrement'.

2.2 Appel de macros

Une fois la macro enregistrée, on peut noter cinq méthodes pour l'exécuter :

- la première, la plus simple à mettre en oeuvre, est d'aller dans le **menu Macro** dans **Outils**, de choisir notre macro, et de cliquer sur **Exécuter**. L'inconvénient de cette méthode est évidemment qu'elle devient longue et complexe dès que le nombre de macro devient important. De plus, il ne sera pas simple pour l'utilisateur final, si ce n'est pas vous, de s'y retrouver.
- Deuxième méthode : le **raccourci-clavier**. L'association d'une macro à un raccourci se fait, soit à l'enregistrement de la macro, soit, dans le menu **Macro**, dans **Options**. Dans les deux cas, on a une case précédée de "CTRL + ". Indiquez dans cette case la combinaison de touches qui vous intéresse. Il y aura donc toujours la touche 'CTRL', mais vous pouvez

ajoutez une lettre avec en plus l'une ou l'autre des touches 'ALT' ou 'MAJ' (en les appuyant simultanément à votre lettre). Dans Excel, il suffira de reproduire cette combinaison de touche pour exécuter votre macro. Ceci peut être très pratique, notamment pour des macros qui effectuent de la mise en forme du texte.

- Troisième méthode, peut-être la plus pratique, est d'utiliser un **bouton**. Utilisez la barre d'outils 'Formulaires' et allez cliquer sur le 'bouton'. Dessinez votre bouton. Excel vous demande alors à quelle macro il doit être affecté. Vous pouvez ensuite changer le texte de votre bouton. Il suffira ensuite de cliquer sur le bouton pour exécuter la macro.
- Nous pouvons également associer une macro à une **image**. Insérez une image comme vous le feriez habituellement. Avec le bouton droit, cliquez dessus et sélectionnez 'Affecter à une macro'. Choisissez votre macro. Un simple clique sur l'image exécutera désormais votre macro.
- Une méthode qui peut s'avérer pratique lors d'application volumineuse est d'ajouter des nouveaux boutons dans les **barres d'outils**. Pour cela, ajoutez un nouveau bouton (cliquez sur la petite flèche à droite d'une de vos barres d'outils) et sélectionnez le bouton 'Macro' (un smiley). Un nouveau bouton s'affiche. Il suffit de cliquer dessus pour qu'il vous demande à quelle macro l'associer. Les prochaines utilisations du bouton appelleront la macro.

En pratique

Cinq appels de macros possibles :

- par le menu Outils / Macro / Exécuter,
- par raccourci clavier,
- par un bouton de la barre d'outils formulaire,
- par une image,
- par un bouton de barre d'outils.

2.3 Modification des macros

Une fois qu'une macro a été enregistrée, elle peut être lue dans Visual Basic Editor pour être modifiée. Pour ceci, allez dans le menu Outils, puis Macro, sélectionnez votre macro, et cliquez sur Modifier.

Dès lors, dès qu'une fonction de VBA vous est inconnue, il est possible de la retrouver simplement en l'enregistrant avec l'enregistreur de macro, et d'examiner le code produit.

3 Algorithmique

3.1 Écriture d'une macro

Une macro VBA se présente sous la forme suivante :

```
Sub nom_de_la_macro()  
  ' macro polie  
  msgbox "Bonjour !"  
End Sub
```

- La macro commence par le mot clé Sub. Ce mot clé est suivi du nom de la macro (un mot sans espace) suivi d'un couple de parenthèse. On notera que le nom d'une macro suit

nécessairement les règles suivantes : il ne contient que des lettres, des chiffres, et certains caractères spéciaux (comme le souligné). Il ne contient aucun espace et doit commencer par une lettre.

- Suit le *code* de la macro, c'est-à-dire la suite d'instructions correspondant à la macro. Ici, la seule ligne correspondant à du code est `msgbox "Bonjour !"` qui permet d'afficher un message à l'utilisateur.
- Il est possible de glisser parmi ce code des *commentaires* : il s'agit de texte placé après le symbole `' '` (comme ici `' macro polie`. Ce texte qui ne sera pas pris en compte lors de l'exécution de la macro. Il peut être très utile (et même fortement recommandé) de glisser des commentaires dans votre macro pour indiquer ce que la macro est censée réaliser ou pour permettre de comprendre facilement des parties de code un peu "difficiles". Ainsi si vous ou une autre personne devait modifier la macro par la suite, le travail en sera d'autant simplifié ;
- enfin, la macro termine par les mots-clés `End Sub`.

3.2 Instructions, séquence et msgbox

Comme nous venons de le voir, les macros sont constitués d'instructions. Une instruction est tout simplement un ordre que l'ordinateur devra exécuter. Voyons tout d'abord une première instruction :

- `msgbox message` : permet d'afficher un message dans une boîte de dialogue. Par exemple `msgbox "hello !"`

On dit que les instructions d'une macro sont placées *en séquence*, c'est-à-dire les unes à la suite des autres. C'est ainsi qu'on les écrit, mais c'est également aussi que l'ordinateur les exécute. Par exemple :

```
Sub bonjour_les_amis()  
  ' affiche des messages polis  
  msgbox "Bonjour !"  
  msgbox "Au revoir !"  
End Sub
```

Lorsque l'on exécute cette macro, l'ordinateur commencera par afficher "Bonjour !" et ensuite "Au revoir !". Il exécute bien les deux instructions dans l'ordre.

3.3 Variables et InputBox

Les variables constituent des éléments essentiels de la programmation. Elle permet de mémoriser des informations. Chaque variable peut ainsi être vue comme une 'case' dans laquelle on pourra mettre *Une* information (un nombre, un texte...). On peut considérer une variable comme une mémoire d'une calculette ou encore comme une case d'un tableau Excel : le fonctionnement en est très similaire.

Chaque variable est identifiée par un *nom*. Ce nom doit respecter les mêmes règles que les noms de macro : il ne contient que des lettres, des chiffres, et certains caractères spéciaux (comme le souligné). Il ne contient aucun espace et doit commencer par une lettre.

On peut ensuite réaliser deux opérations élémentaires avec une variable : y stocker une valeur ou récupérer la valeur qui y est stockée.

- Pour stocker une valeur dans une variable, on utilise le signe égal "=" (appelé opérateur d'affectation). On écrit d'abord le nom de la variable, puis "=" et enfin la valeur que l'on souhaite stocker dans la variable. Cette valeur peut être issue d'un calcul, auquel cas l'ordinateur effectuera le calcul et stockera le résultat dans la variable. Par exemple :

```

a = 2          ' stocke la valeur 2 dans la variable 'a'
nom = "toto"   ' stocke la valeur "toto" dans la variable nom
a = 3 * 5 + 1 ' stocke la valeur 16 dans la variable 'a'
              ' (la variable 'a' oublie donc la valeur 2)

```

Attention ! le signe égal employé ici n'est pas le même signe égal qu'en mathématiques et une variable n'est pas non plus une inconnue mathématique. Le fait d'écrire `a = 2` ne signifie pas que la variable 'a' vaut définitivement '2'. On ne fait que mettre la valeur '2' dans la variable 'a'.

- pour récupérer la valeur d'un variable, indiquez simplement son nom. L'ordinateur remplacera automatiquement la variable par son contenu. Par exemple :

```

nom = "toto" ' on stocke "toto" dans la variable 'nom'
msgbox nom   ' nom est remplacé par "toto", on affiche donc "toto"
a = 2        ' stocke 2 dans a
b = 3 * a    ' a est remplacé par 2, on stocke donc 6 dans a
a = a + 1    ' on commence par effectuer le calcul de droite qui vaut 3,
              ' la nouvelle valeur de 'a' est donc 3

```

On notera également qu'on peut utiliser l'instruction suivante pour donner une valeur à une variable :

- `inputbox(message)` : affiche une boîte de dialogue avec le message indiqué et une zone de saisie. La valeur entrée par l'utilisateur est ensuite retournée et peut donc être stocké dans une variable. Par exemple :

```
nom = inputbox("quel est votre nom ?")
```

Ceci permet de demander à l'utilisateur son nom et de stocker sa réponse dans la variable 'nom'.

3.4 le branchement conditionnel (IF)

Il est possible de construire des macros avec des structures plus élaborées que de simples séquences. On utilise pour cela des *structures de contrôle*. La première de ces structures de contrôle est la structure conditionnelle. Vous la connaissez déjà : en Excel, elle s'appelle "SI" dans Excel.

Cette structure est notée en VBA `IF ... THEN ... ELSE ... END IF`. On l'écrit de la manière suivante :

```

if (condition) then
  instructions 1
  ...
else
  instructions 2
  ...
end if

```

Une condition est un *test logique*. Pour l'instant, nous n'utiliserons qu'une seule condition `a = 1` qui vérifie si la variable a contient la valeur 1. Nous verrons dans la section suivante d'autres formes de conditions.

Tout d'abord, la condition est testée. Si cette condition est remplie, l'ordinateur exécute la première séquence d'instructions, sinon, il exécute la deuxième séquence (celle après le `else`). Par exemple :

```

if (a = 1) then
  msgbox "bonjour"
  b = 2
else
  msgbox "Au revoir"
  b = 3
end if

```

On commence par vérifier que la valeur 1 est bien stockée dans la variable 'a'. **Si** c'est le cas, **alors** on écrit "bonjour" puis on stocke 2 dans la variable 'b'. **Sinon** on écrit "Au revoir" et on stocke 3 dans la variable 'b'.

Notons qu'il est courant qu'il n'y ait rien à effectuer dans le cas où la condition n'est pas remplie. On peut alors se passer de la partie **else**. On écrira alors :

```

if (condition) then
  instructions

  ...
end if

```

3.5 Conditions et tests logiques

Une condition, telle qu'employé par le IF, est une combinaison d'un ou de plusieurs tests logiques simple. Un test logique élémentaire est tout simplement un test dont la valeur est 'oui' ou 'non'. Le test logique le plus courant est la comparaison qui s'écrit de la manière suivante :

- (**expression1 comparateur expression2**). Expression1 et Expression2 sont soit des nombres, soit des variables, soit des calculs plus complexes. Le comparateur est usuellement l'un des comparateurs suivants : = (égalité) < (plus petit que), > (plus grand que), <= (plus petit ou égal), >= (plus grand ou égal) et <> (est différent de). Le résultat du test est vrai ou faux, ou plus exactement *true* ou *false*.

Par exemple :

```

(1 = 2)      ' vérifie si 1 et 2 sont identiques, vaut donc false
(a = 3)      ' vérifie si a contient 3
(3 = a)      ' idem
(a + 2 <> b * 2) ' vaut vrai seulement si a+2 est différent du double de b

```

Attention : l'opérateur d'égalité est encore une fois différent de l'égalité mathématique, mais également de l'opérateur d'affectation (noté également '=') vu plus haut. Il indique juste vrai si le résultat du calcul de l'expression de gauche est le même que le résultat du calcul de l'expression de droite. Il a d'ailleurs la même signification que le signe '=' que vous emploieriez dans une formule Excel classique.

Comme dans Excel, il est possible de réaliser des conditions plus complexes en combinant plusieurs tests simples avec des *opérateurs logiques*. On utilise surtout les opérateurs suivants :

- **not** : le 'non', ou l'inversion. 'Inverse' un test logique simple. Par exemple **not(a < 5)** est vrai si a est supérieur ou égal à 5.
- **and** : le 'et' ou la conjonction. Combine deux tests logiques, vaut vrai si les deux tests étaient vrai, faux dans tous les autres cas. Par exemple **(a >= 5) and (a <= 10)** est vrai si a est supérieur ou égal à 5 **et** si a est inférieur ou égal à 10 (soit si a est entre 5 et 10 inclus).
- **or** : le 'ou' ou la disjonction. Combine deux tests logiques et vaut vrai si au moins l'un des tests est vrai. Par exemple, **(a < 10) ou (b < 10)** est vrai si a est inférieur à 10 **ou** si b est inférieur à 10.

3.6 La boucle for

Il arrive souvent d'avoir besoin de réaliser plusieurs fois la même séquence d'instructions. Les *boucles* permettent ceci. Nous voyons tout d'abord ici la boucle 'for'. Cette boucle permet de répéter un nombre choisi de fois une séquence d'instruction. Ceci se fait en utilisant une variable appelée *compteur* qui permet de compter le nombre de répétition. La syntaxe est la suivante :

```
for (variable) = (début) to (fin)
  instructions
next
```

Il faut remplacer (variable) par le nom de la variable choisie comme compteur, (début) pour la valeur de début et (fin) pour la valeur de fin. La séquence d'instruction est alors exécutée une première fois avec la valeur de début stockée dans le compteur, puis une deuxième fois avec (début)+1 dans le compteur et ainsi de suite jusqu'à ce que la valeur (fin) soit atteinte. Par exemple :

```
for i = 10 to 20
  msgbox i
next
```

ceci s'exécute de la manière suivante :

- tout d'abord, i contient la valeur '10'. `msgbox i` affiche donc 10.
- On répète ensuite avec la valeur 11 dans 'i', et on affiche 11.
- les répétitions suivantes affichent 12, puis 13. On continue ainsi de compter jusqu'à ce que la valeur 20 soit assignée à i. On affiche alors 20, et la boucle est terminée.

La boucle `for` s'utilise souvent pour répéter un nombre fixé de fois une séquence d'instruction. Dans ce cas, le compteur ne servira probablement pas dans cette séquence. Ainsi, le code suivant écrira "bonjour" un nombre de fois égal au contenu de la variable 'n'

```
for i = 1 to n
  msgbox "bonjour"
next
```

3.7 La boucle do ... loop

La boucle Do ... Loop est la boucle la plus générique qui soit. Sa structure de base est la suivante :

```
Do [while|until] (condition)
  Instructions
  ...
Loop
```

La séquence d'instructions qui est indiquée entre le 'do' et le 'loop' est répétée. Reste à indiquer jusqu'à quel moment. Ceci se fait à l'aide du mot clef 'while' ou 'until'. While (Tant Que) suivi d'une condition permet de faire continuer la répétition tant que la condition est vraie. Until (Jusqu'à) fait l'inverse : la répétition se fait jusqu'à ce que la condition soit vraie. On place while/until juste après 'Do' (notons qu'il est également possible de le placer après le 'Loop' avec un fonctionnement légèrement différent - voir l'aide en ligne pour plus de détail).

Par exemple, pour faire un compte à rebours, on peut faire :

```

i = 10          ' on stocke 10 dans la variable i
do until i = 0
  msgbox i     ' on affiche la valeur stockée dans i
  i = i - 1    ' on ôte 1 de i et on stocke le résultat dans i
loop
msgbox "fini"

```

Ceci s'exécute de la manière suivante :

- on stocke 10 dans la variable i ;
- on rentre dans la boucle ;
- on affiche la valeur stockée dans i, soit 10 ;
- le calcul de la partie droite donne 9, on stocke donc 9 dans i ;
- le test de la boucle s'effectue : i ne vaut pas 0, on recommence donc ;
- on affiche la valeur stockée dans i, soit 9 ;
- on continue ainsi jusqu'à arriver à la valeur de 1 dans i ;
- on affiche i (1), puis on stocke 0 dans i ;
- on réalise le test final : i vaut bien 0, on sort donc de la boucle ;
- on affiche "fini".

On pourra noter que la boucle `do ... loop` est plus générale que la boucle `for ... next`. Ainsi, les deux structures suivantes produisent le même résultat.

```

for i = 1 to 10
  msgbox i
next

i = 1
do
  msgbox i
  i = i + 1
loop while i <= 10

```

3.8 Branchement conditionnel : select case

Une dernière structure de données que nous présentons ici est le branchement conditionnel. Cette structure est en quelque sort un 'if' évolué. Il permet en quelque sortes de faire plusieurs 'if' en même temps. Il est courant en effet d'avoir des macros dont le comportement varie en fonction de la valeur d'une variable. Supposons que la variable 'civilite' contienne 'Mr', 'Mlle' ou 'Mme' et qu'on désire afficher respectivement 'Monsieur', 'Mademoiselle' ou 'Madame' selon le cas. Il y a donc trois valeurs possibles, et trois comportements différents. Cela s'effectuerait de la manière suivante à l'aide de 'if' :

```

if (civilite = "Mr") then
  msgbox "Monsieur"
end if
if (civilite = "Mme") then
  msgbox "Madame"
end if
if (civilite = "Mlle") then
  msgbox "Mademoiselle"
end if

```

Il y a moyen de faire plus court avec le `select case`. La syntaxe est la suivante :

```
select case (expression) ' expression est la valeur à tester
  case valeur1
    instructions1        ' séquence à réaliser si la valeur
    ...                  ' de l'expression est valeur1
  case valeur2
    instructions1        ' séquence à réaliser si la valeur
    ...                  ' de l'expression est valeur2
  case Else
    instructions1        ' séquence à réaliser si la valeur
    ...                  ' de l'expression est valeur1
end select
```

Le cas précédent s'écrit ainsi :

```
select case (civilite)
  case "Mr"
    msgbox "Monsieur"
  case "Mme"
    msgbox "Madame"
  case "Mlle"
    msgbox "Mademoiselle"
end select
```

Il est également possible de faire des tests plus complexes. A la place de mettre une simple valeur après le `case`, on peut également mettre plusieurs valeurs (ex : 1, 5, 6), une plage de valeur (ex : 1 to 5) ou un opérateur de comparaison précédé de `Is` et suivi d'une valeur (ex : `Is >= 5`). On pourra ainsi écrire :

```
select case (age)
  case 0,1
    msgbox "bebe"
  case 2 to 12
    msgbox "enfant"
  case 13 to 17
    msgbox "adolescent"
  case Is >= 18
    msgbox "adulte"
end select
```

4 Procédures et fonctions

4.1 Procédures

Nous voyons maintenant plus en détail la structure générale des macros. Les macros que nous avons vu jusqu'à présent sont également appelées des procédures. Nous avons vu qu'elles étaient délimitées par les mots clefs `Sub` et `End Sub`. Nous avons vu qu'une procédure peut être exécutée de diverses manières (par un bouton par exemple).

Une procédure est avant tout un ensemble d'instructions qu'il est possible d'exécuter à n'importe quel moment, y compris à partir d'une autre procédure. Voyons par exemple la procédure simple suivante :

```
Sub macroBonjour()
  msgbox "bonjour les amis !"
End Sub
```

Si dans une autre macro, nous avons besoin d'afficher "bonjour les amis !", il n'est plus nécessaire de réécrire ce code : on peut à la place faire appel à la macro `macroBonjour`. Ceci se fait simplement en écrivant le nom de la macro. Par exemple :

```
Sub macroTest()  
    nom = inputbox("Quel est votre nom ?")  
    macroBonjour  
End Sub
```

Cette macro commence par demander son nom à l'utilisateur pour le stocker dans la variable `nom`, puis exécute le contenu de la macro `macroBonjour`.

La pratique qui consiste à regrouper réaliser de nombreuses procédures est en général une bonne pratique et ce pour les raisons suivantes :

- lisibilité du code : si chaque procédure réalise une tâche précise, il est alors plus simple de comprendre ce qu'elles font, ceci aide grandement lorsqu'il faut modifier ou corriger les macros,
- code plus court : si deux macros ont une partie similaire, le fait de regrouper ces parties similaires dans une troisième macro fait un code plus court, et par conséquent plus lisible. Cela permet également d'écrire son code plus vite et de limiter les erreurs par copier-coller.

4.2 Passage d'arguments

Il est possible de paramétrer les procédures, c'est-à-dire d'envoyer certaines valeurs à une macro pour faire varier son comportement. Par exemple, supposons que nous désirions écrire une fonction qui affiche "Bonjour, " suivi du nom d'une personne.

Ceci s'écrit de la manière suivante :

- du côté de la procédure à paramétrer, on indique sur la première ligne (entre les parenthèses) une liste de variables dont chacune correspond à un paramètre.
- au moment d'appeler la procédure, on indique après le nom de la procédure la liste des valeurs qui seront utilisés comme arguments.

Par exemple :

```
Sub bonjour (nom)  
    msgbox "Bonjour " & nom  
End Sub
```

La procédure `bonjour` a un argument : `'nom'`. Lors de l'exécution de la procédure, la variable `nom` contiendra la valeur de l'argument, comme dans l'exemple suivant :

```
sub test()  
    bonjour "Bob"  
    bonjour "Sam"  
End Sub
```

Ici, la procédure `bonjour` est appelée deux fois :

- une fois avec l'argument `Bob` : la variable `'nom'` de la procédure contient, au moment du premier appel de "bonjour", la valeur `'bob'`
- la deuxième fois, la variable `nom` contiendra `'Sam'`.

5 Fonctionnalités Excel

5.1 Les objets Excel

La manipulation d'Excel se fait à travers la notion d'objet. Tout élément d'Excel est un objet : une feuille de calcul, une cellule, un bouton, un graphique... L'utilisation des macros sous excel passe donc par la manipulation de ces objets.

La première étape pour manipuler un objet sous Excel consiste à identifier cet objet. Ainsi, un certain nombre de mots clefs permettent de sélectionner les divers objets d'Excel. Par exemple, la cellule 'active' se nomme 'activecell'. La feuille active s'appelle 'activesheet'.

La première manière d'agir sur un objet Excel est de lui donner un ordre (que nous appellerons ici une *méthode*). A chaque type d'objet est associé un ensemble de méthodes. Par exemple, la méthode 'clear' peut effacer le contenu d'une cellule.

Pour utiliser une méthode, on indique d'abord l'objet auquel on donne un ordre, on met ensuite un point, puis la méthode. Par exemple `activecell.clear` efface la cellule courante.

Chaque objet a également tout un ensemble de propriétés, de caractéristiques. Par exemple, une cellule est caractérisée par ses dimensions, sa couleur, son contenu... Chacun de ses éléments est appelée une propriété. Pour sélectionner une propriété, on utilise la même technique que pour les méthodes : on indique tout d'abord l'objet sur lequel on travaille, on met un point, et on indique ensuite le nom de la propriété. Par exemple, la propriété `value` d'une cellule correspond à la valeur qu'elle contient. Ainsi, `activecell.value` correspond à la valeur de la cellule courante.

Notons qu'une propriété est en réalité une variable. On peut donc, comme une variable, récupérer ou changer sa valeur. Par exemple :

```
Sub machin()  
activecell.value = "toto"    ' on place le mot "toto" dans la cellule active  
msgbox activecell.value    ' on affiche le contenu de la cellule active  
End Sub
```

5.2 Objets de type "cellule"

Nous allons en réalité utiliser essentiellement les objets de type 'Cellule', puisqu'ils constituent la base de nos feuilles Excel. Pour indiquer la cellule qui nous intéresse, trois possibilités nous sont offertes :

- **activecell** est la cellule *active*, c'est-à-dire celle qui est sélectionnée,
- **range** nous permet de sélectionner une cellule par son nom, tel qu'il apparaît dans Excel. Ainsi `range("C3")` nous permet de sélectionner la cellule de coordonnées C3.
- **cells** réalise la même opération, mais en indiquant les coordonnées de la cellule (attention ! dans l'ordre ordonnée puis abscisse). Ainsi `cells(3,5)` est la cellule E3.

A cela, il faut ajouter la possibilité de sélectionner une cellule sur une autre feuille en précisant le nom de la feuille avec le mot clé **sheets**. Ainsi `sheets("feuille1").range("C3")` est la cellule C3 de la feuille `feuille1`.

Pour manipuler les cellules, nous voyons ici deux méthodes. Il en existe évidemment beaucoup d'autres :

- **clear** : efface la cellule. Par exemple, `range("A1").clear` efface le contenu de la cellule A1.
- **select** : sélectionne la cellule. Cela revient à cliquer sur la cellule en quelque sorte. La cellule devient alors la cellule active. Ainsi `cells(1,5).select` indique que la cellule A5 devient sélectionnée.

De même, nous voyons une propriété des cellules (parmi beaucoup d'autres) :

- **Value** : cette propriété est tout simplement la valeur, le contenu de la cellule. Ainsi `activecell.value` est le contenu de la cellule active.

Par exemple, le code suivant permet de placer dans la cellule B2 la valeur de la cellule B1 :

```
Sub test()  
    range("B2").value = range("B1").value  
End sub
```

6 Un exemple de macro : Parcours de liste

Pour réaliser un parcours de liste, il nous faut voir une technique qui nous permet de repérer une cellule à partir d'une autre : le mot clé **offset**. La syntaxe est la suivante : `(cellule de départ).offset(décalage Y, décalage X)`. Par exemple, la cellule `activecell.offset(1,0)` est la cellule en dessous de la cellule courante, `activecell.offset(0,-1)` est la cellule à gauche de la cellule courante.

Si on veut, par exemple, rechercher la première cellule de la colonne A qui contient le mot 'truc', on peut utiliser le code suivant :

```
Sub chercheTruc()  
range("A1").select  
do while ((activecell.value <> "truc") or (activecell.value <> ""))  
    activecell.offset(1,0).select  
loop  
End Sub
```

Le déroulement de cette macro est le suivant :

- on sélectionne la cellule A1 : elle devient la "cellule courante"
- on va réaliser la recherche tant qu'on n'a pas trouvé "truc" et tant que nous ne sommes pas arrivé en bas de la liste. Pour l'instant, ce n'est pas le cas.
- la ligne `activecell.offset(1,0).select` sélectionne la cellule en dessous de la cellule active. En gros, on fait descendre la sélection d'une case
- et on boucle donc, jusqu'à avoir trouvé "truc", ou que la fin de la liste soit atteinte.

Pour résumer, on indique qu'on commence par sélectionner la cellule "A1", et on va faire descendre la sélection jusqu'à atteindre la cellule qui contient l'information intéressante, ou jusqu'à atteindre la fin de la liste (c'est-à-dire une cellule vide).