Introduction aux Macros et à Visual Basic pour Applications

– Olivier Losson –

Note préliminaire
Il est surtout fait référence ici aux interfaces, commandes et fonctions de Word, Excel et Powerpoint (celles d'Access en diffèrent sensiblement).

1. Généralités et définitions

Dans une application Microsoft de la suite Office (Excel, Word, Access ou PowerPoint), chaque fois qu'une même séquence d'actions doit être effectuée périodiquement, il est possible d'enregistrer celles-ci dans une macro-commande de manière à les réaliser ensuite automatiquement, autant de fois qu'on le désire. Une macro-commande (***macro*** en abrégé) est donc une suite de commandes et permet d'automatiser certaines tâches que l'on est amené à effectuer de manière répétitive, éventuellement avec des données différentes (ex. : mise en forme d'un tableau, recherche ou formatage de données, …).

Avec les macros, on peut créer des commandes complexes, de nouvelles fonctions, des interfaces graphiques avec menus, boîtes de dialogue, boutons personnalisés, … En un mot, les macros permettent à chacun de créer des outils adaptés à ses propres besoins. Avant de créer une macro pour automatiser une tâche, on doit toutefois s'assurer qu'aucune solution intégrée n'est fournie. Par exemple, si on veut sélectionner toutes les cellules vides d'une feuille Excel, il suffit de choisir le menu *Edition/Atteindre* et de cliquer sur *Cellules…* dans la boîte de dialogue.

Une macro est un "programme" qui exécute une suite de tâches bien définies, grâce à des instructions écrites :

- jusqu'à récemment (version 5 d'Excel, 6 de Word), dans un ***langage macro*** ; le programme était alors édité dans un document classique (feuille de calcul ou document Word par exemple) ;
- actuellement en ***Visual Basic pour Applications (VBA)***, qui est un sous-ensemble du langage de programmation Visual Basic, et possède de nombreux avantages :
  - *Portabilité*. La macro est éditée dans une fenêtre dédiée (l'***éditeur***) dont la présentation est la même pour toutes les applications Office. De plus, les macros sont stockées dans des ***modules***, ce qui permet de les retrouver et de les réutiliser aisément.
  - *Orientation objet*. Les éléments manipulés sont spécifiques à chaque application ***hôte*** : il s'agit par exemple de blocs de texte sous Word, de plages de cellules sous Excel ou de données stockées dans une table sous Access. Appelée ***objets,*** ceux-ci possèdent certaines caractéristiques bien définies, ainsi qu'un comportement propre (on dit qu'ils sont *encapsulés*). Ainsi, toute commande que vous pourriez réaliser sur un objet (ex. suppression d'un bloc de texte, formatage d'une plage de cellules ou édition d'un enregistrement d'une table) possèdent un équivalent sous forme de code VBA (*procédure*). L'ensemble des objets d'une application est regroupé dans une *bibliothèque*.
  - *Fonctionnalités intégrées* de débogage et possibilité de compiler les modules de code en macros complémentaires distribuables pour réaliser des applications professionnelles sécurisées.

Par ailleurs, les applications Office contiennent un ***enregistreur*** de macros, outil intégré qui crée le code VBA à notre place.

2 – Enregistrement et exécution de macros

2.1. Enregistrement

Si la macro à créer est simple, ou pour générer rapidement une première ébauche d'une macro plus complexe, on peut utiliser l'enregistreur de macros. Celui-ci fonctionne comme un magnétoscope : il mémorise les actions que vous effectuez (sélections, commandes, …) dans un module Visual Basic pour pouvoir les reproduire le moment voulu.

L'enregistreur est accessible à partir du menu *Outils/Macro/Nouvelle macro…* On donne alors le nom de la macro[1], éventuellement une touche de raccourci et une description, ainsi qu'un ***document de stockage***[2].

Lorsqu'on appuie sur OK, l'enregistrement commence : toutes les actions que vous effectuez alors sont stockées dans le code de la macro et seront reproduites à l'identique lors de son exécution. Pour terminer l'enregistrement, appuyer sur le bouton  (ou *Outils/Macro/Arrêter l'enregistrement*).

2.2. Exécution et modification

Pour exécuter une macro, ou gérer les différentes macros attachées au document (modification, suppression, …), utiliser la commande *Outils/Macro/Macros…* En appuyant sur l'un des boutons *Créer* ou *Modifier* de la même boîte de dialogue, on accède à l'éditeur Visual Basic pour, respectivement, créer de toutes pièces ou modifier une macro enregistrée.

3 – L'éditeur Visual Basic (VBE)

3.1. Vue générale

A partir de la version 97 de la suite Office, la présentation de l'éditeur VBA a été homogénéisée (cf. figure 1 page suivante) ; il se présente dans une fenêtre distincte de celle de l'application hôte, ce qui permet de séparer le code des données.

3.1.1. Fenêtre Explorateur de Projet

Un ***projet*** regroupe l'ensemble des documents (i.e., sous Excel, les feuilles de calcul associées à un classeur, et sous Word, le document texte) et des macros. A partir de la racine de l'arborescence du projet, les documents sont accessibles dans la branche *Microsoft Excel|Word|PowerPoint Objects* et les macros dans la branche *Modules.*

---

[1] Le nom de la macro doit commencer par une lettre, ne pas comporter d'espace ni de caractère de ponctuation, et ne peut être un mot réservé de VBA ; éviter également les caractères accentués.

[2] Une macro est liée à un document qui la contient ; elle ne peut être exécutée **que si ce dernier est ouvert**. La macro peut être enregistrée avec le document courant, mais pour la rendre accessible en permanence, il est aussi possible de la placer :
- Sous Word, dans le modèle global *NORMAL.DOT*.
- Sous Excel, dans un classeur de macros personnelles, nommé *PERSO.XLS*, et placé dans le répertoire *XLOUVRIR* pour qu'il soit ouvert à chaque démarrage d'Excel (il reste cependant invisible ; activer le menu *Fenêtre/afficher* pour pouvoir modifier les macros qu'il contient).
- Sous Powerpoint, cela est impossible (la macro n'est accessible que de la présentation où elle est enregistrée).

t En phase de mise au point, stockez d'abord vos macros dans le document courant et transférez-les dans la bibliothèque de macros personnelles lorsqu'elles sont au point.

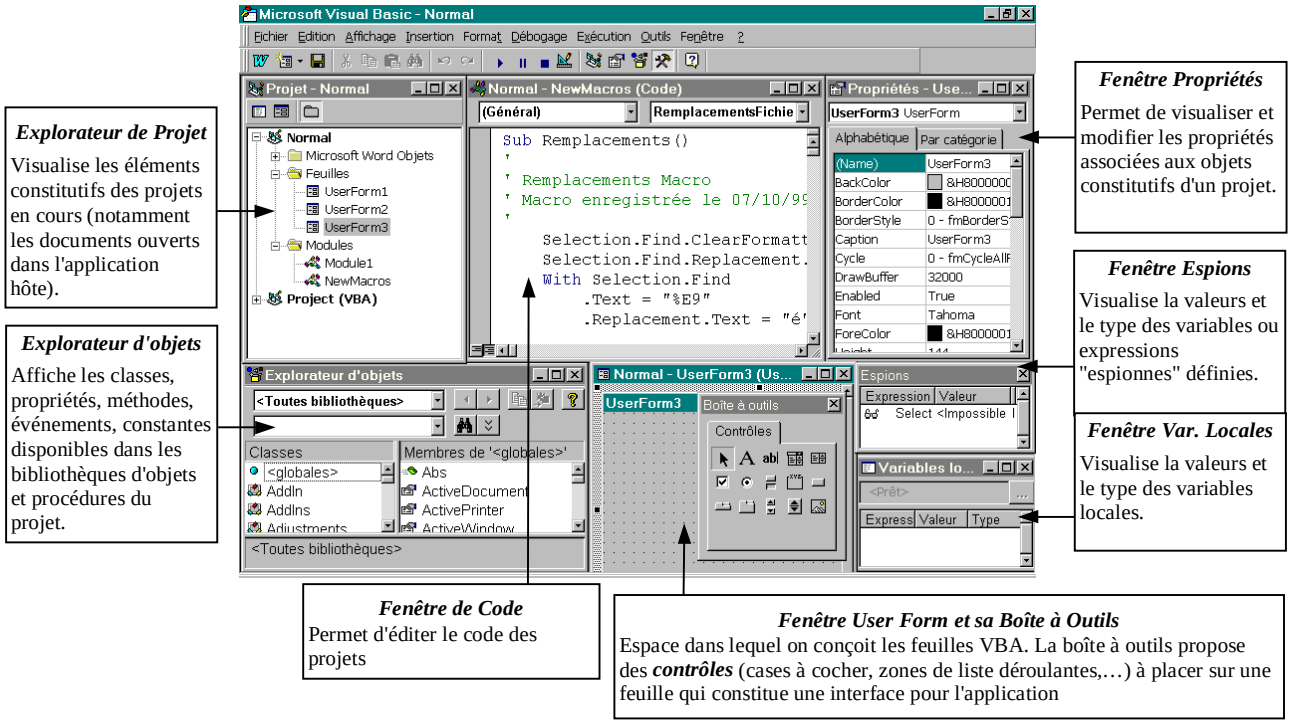


Figure 1 : Fenêtre de l'éditeur VBA

3.1.2. Fenêtre de Code

C'est ici que vous modifiez le code généré. En général, l'enregistreur de macros génère plus de code que nécessaire ; il faut donc le « dépoussiérer » en retirant les instructions et options de commandes superflues afin de minimiser le temps d'exécution de la macro.

Dans cette fenêtre, les mots-clés réservés apparaissent en bleu et les commentaires en vert (ceci est personnalisable dans *Outils/Options/Format de l'éditeur*).

3.1.3. Fenêtres Espions et Variables locales

Voir la section « 7. Exécution pas à pas et débogage » ci-après.

3.2. Aides à l'édition

Outre l'enregistreur de macros, qui permet souvent de générer une première ébauche de votre programme, vous disposez sous VBE de plusieurs systèmes d'aide à l'édition de code :

- *vérification automatique de la syntaxe* : si cette option est active, lors du passage à la ligne suivante, l'éditeur signale les erreurs syntaxiques les plus évidentes (par ex. s'il manque *then* après *if*) et met en majuscules les objets, propriétés et méthodes qu'il connaît.

Tapez donc tout en minuscules; si votre code est correct, il sera formaté automatiquement.

- *aide contextuelle* : placer le curseur dans un mot, ou sélectionner celui-ci, puis appuyer sur *F1* ;
- *aide classique*, avec son sommaire et ses fonctions de recherche. La rubrique *Objets Microsoft Word | Excel | PowerPoint…* permet notamment de parcourir la hiérarchie des classes d'objets ;
- *affichage des propriétés et méthodes applicables* à un objet lorsque l'on tape le "**.**" de séparation ;
- *explorateur d'objets* : affiche les membres (propriétés, méthodes et événements) des objets (cf. plus loin pour ces différentes notions).

4 – Notions de Module et de Procédure

4.1. Procédure

4.1.1. Définition

En fait, le mot *macro* a été conservé par « compatibilité de vocabulaire » avec les versions antérieures d'Office. Mais dorénavant, le code VBA est organisé en ***procédures***, ce qui le rend plus performant et lisible. Les mots *macro* et *procédure* désignent donc la même notion et seront employés indifféremment.

Une macro débute par le mot *Sub* (abréviation de l'anglais *Subroutine* = procédure), suivi du nom de la macro, et se termine par les mots *End Sub*. Une procédure est une suite d'*instructions*, chacune exécutant une tâche précise.

4.1.2. Relation entre code généré et procédure Sub

Lors de l'enregistrement d'une macro est générée une procédure *Sub* portant le nom spécifié. Il est possible de renommer celle-ci directement dans la fenêtre de code, ou bien dans l'application hôte. De même, la suppression des lignes comprises entre la ligne d'en-tête *Sub* et la fin de la macro repérée par *End Sub* équivaut à choisir dans l'application le menu *Outils/Macro/Macros…* puis *Supprimer*. L'exécution d'une macro peut également se lancer à partir de l'éditeur en appuyant sur le bouton  , le curseur étant positionné dans le code correspondant.

4.1.3. Divers types de procédures

On peut définir deux types de macros :

- les macros ***commandes*** (de type *Sub*) qui accomplissent un ensemble d'actions mais ne retournent pas de valeur [1] :

```
[Private|Public] [static] Sub nomProcedure ([arguments])
    Instructions
End Sub
```

- les macros ***fonctions*** (de type *Function*) qui retournent une valeur :

```
[Private|Public] [Static] Function nomProcedure ([arguments]) [As Type]
    Instructions
    nomProcedure = expression                    'Valeur retournée par la fonction
End Function
```

Le nom *nomProcedure* peut ensuite être utilisé pour appeler la procédure à partir d'autres procédures, en respectant les arguments requis. L'instruction *Call* permet d'appeler une procédure *Sub*. Pour la déclaration des arguments d'une procédure (passage par valeur ou par adresse, types des arguments, arguments optionnels, …), voir dans l'aide la rubrique correspondant à l'instruction *Sub* ou *Function* ; pour l'utilisation ou non des parenthèses, voir « *Appel de procédures Sub et Function* ».

4.2. Module

Les macros d'un projet sont stockées dans des ***modules***, un module pouvant contenir plusieurs macros. Un module peut être sauvegardé individuellement (menu *Fichier/Exporter un fichier*) dans un fichier d'extension *.BAS* afin de le réintégrer dans un autre projet (grâce à *Fichier/Importer un fichier*).

[1] L'enregistreur génère uniquement ce type de macros, sans argument.

5 – Notions liées aux objets

5.1. Notion d'Objet

VBA est un langage ***orienté objets*** : tous les éléments de l'application hôte sont des *objets* (ex. sous Excel : un classeur, une feuille de calcul, une plage de cellules, etc.). Un objet est en réalité un élément spécifique d'une application qui contient à la fois des données et du code.

L'intérêt principal d'un objet est d'offrir au programmeur des composants existants et réutilisables, qu'il lui suffit d'intégrer et d'utiliser tels quels dans ses applications.

5.2. Notion de Propriété et de Méthode

Chaque objet possède certaines caractéristiques (appelées ***propriétés***), qui contrôlent son apparence et son comportement, et peut exécuter un certain nombre d'actions (les ***méthodes***). Par exemple, la sélection courante possède la propriété *Characters* (les caractères englobés dans la sélection) et peut, entre autres, se voir appliquer la méthode *Copy* (pour copier la sélection dans le presse-papiers). Autre exemple, on peut vérifier l'orthographe d'une plage de caractères dans Word ou de cellules dans Excel, en appliquant la méthode *CheckSpelling* à cet objet.

5.2.1. Lecture et modification des propriétés

Pour connaître la caractéristique d'un objet, il faut accéder à la valeur de la propriété correspondante, ce qui est réalisé en faisant suivre la référence à cet objet d'un "**.**" et du nom de la propriété. Ainsi, pour récupérer la couleur des caractères de la sélection courante sous Word :

```
Selection.Font.ColorIndex
```

Dans cet exemple, on accède d'abord à la propriété *Font* de l'objet *Selection*, puis à la propriété *ColorIndex* de l'objet *Font* (comme quelques autres propriétés, *Font* retourne un objet).

Pour modifier la caractéristique d'un objet, il faut accéder à la propriété (comme précédemment), puis affecter une valeur à celle-ci en la faisant suivre d'un signe « égal » et de sa nouvelle valeur. Ainsi, pour affecter la couleur rouge (de code 6) aux caractères de la sélection courante :

```
Selection.Font.ColorIndex = 6
```

Certaines propriétés sont en lecture seule et ne peuvent se voir affecter de valeur (par exemple, les caractères contenus dans la sélection courante *Selection.Characters*).

5.2.1. Exécution de méthodes

Pour déclencher une méthode sur un objet, on utilise également la notation pointée (par exemple *Selection.Copy* pour copier la sélection courante dans le presse-papiers).

Les méthodes ont parfois des ***arguments*** qui en précisent l'exécution. L'exemple ci-dessous expose les deux façons de transmettre des paramètres à une méthode ; il imprime les 3 premières pages du document Word courant :

- passage de paramètres (les virgules sont obligatoires pour séparer les arguments optionnels omis)

```
ActiveDocument.PrintOut , , wdPrintFromTo, , "1", "3"
```

- utilisation d'*arguments nommés*[1] (à utiliser préférentiellement, car plus lisible)

```
Document.PrintOut  From:=1, To:=3
```

[1] On nomme ici les paramètres ; leur valeur est précédée de ":=" et leur ordre est quelconque. Voir les rubriques d'aide « *Appel de procédures Sub et Function* », « *Passage efficace d'arguments* » et « *Arguments nommés et facultatifs* ».

5.3. Notion d'événement

Un ***événement*** est une action de l'utilisateur ou du logiciel qu'il est intéressant de détecter. Il peut s'agir par exemple de la sélection d'une cellule, la saisie d'une valeur, l'ouverture d'un document, l'activation d'un menu, etc. Il est possible d'associer à un événement un fragment de code appelé ***procédure événementielle***. Ce code va s'exécuter lorsque survient l'événement, et permet entre autres de contrôler les manipulations de l'utilisateur ou d'enrichir les commandes intégrées de l'application.

Un événement est associé à un objet donné (que cet objet soit intégré dans l'application ou ait été créé par l'utilisateur ou par programmation). Pour associer une procédure événementielle à un objet, double-cliquer sur celui-ci dans l'explorateur de projet puis, dans la fenêtre de code, choisir dans la liste supérieure droite l'événement à traiter. Le nom de la procédure se présente alors sous la forme :

<Objet>_<Evénement>() par ex. *Worksheet_Activate()* sur activation d'une feuille de calcul

Les annexes présentent les différents événements associés aux objets courants, avec leurs paramètres.

5.4. Notion de collection

Une ***collection*** est un ensemble d'objets connexes (en général du même "type"), appelés ***items***. Ainsi, tous les paragraphes d'un document Word sont contenus dans une seule collection d'objets (*Paragraphs*). Une collection est elle-même un objet, possédant ses propriétés et méthodes propres.

Dans une collection, chaque objet porte un numéro (comme dans un tableau) et un nom (ou ***clé***, comme dans un dictionnaire) permettant d'y accéder. Par exemple, la deuxième feuille d'un classeur Excel (nommée « Feuil2 ») est accessible par *Worksheets(2)* ou *Worksheets("Feuil2")*. Ceci permet d'accéder aux objets et donc, en utilisant la notation pointée comme expliqué précédemment, de modifier leurs propriétés ou de leur appliquer des méthodes (ex. pour fermer le premier document ouvert dans Word, on utilise *Documents(1).Close*).

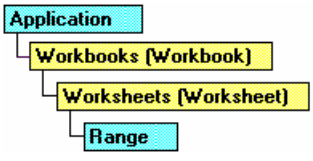
Une collection dispose souvent de méthodes et de propriétés qui peuvent être utilisées pour modifier la totalité des objets qu'elle contient (ex. *Documents.Save* enregistre tous les documents ouverts dans Word).

Remarque : En général, les collections d'objet portent le même nom que le "type" des objets qu'elles contiennent, suffixé de la lettre "**s**" (*Documents* = collection de *Document*).

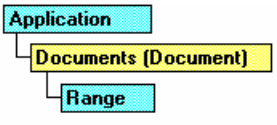
5.5. Hiérarchie des objets

5.5.1. Présentation

Les objets sont classifiés en une organisation hiérarchique de ***classes*** : certains objets en contiennent d'autres (ils sont alors appelés ***conteneurs***). Au sommet de la hiérarchie se trouve l'objet *Application*, qui représente l'application hôte, et constitue le conteneur d'objets le plus vaste. Dans Excel, il englobe notamment des objets *Workbooks*, *CommandBars* et *Dialogs* représentant respectivement des classeurs, des barres de menus et des boîtes de dialogue. Un objet *WorkBook* (un classeur) contient à son tour des objets *Worksheets* et *Charts* représentant respectivement des feuilles de calcul et des feuilles graphiques. Et ainsi de suite … Voici un extrait des classes les plus utiles dans Word et Excel :



Application EXCEL
Classeurs de l'application
Feuilles de calcul de chaque classeur
Plage de cellules (sélection, ligne,…)



Application WORD
Documents de l'application
Zone contiguë de texte

t Les objets peuvent être très nombreux (ex. plus de 100 dans Excel). Pour visualiser le document présentant leur arborescence et accéder ainsi graphiquement aux différents objets, procéder ainsi :
- dans l'éditeur, placer le curseur dans un nom d'objet (ex. *Application*) et appuyer sur F1.
- dans l'aide, rechercher puis afficher la rubrique « *Objets Microsoft Word|Excel|Poxerpoint* », au besoin en activant l'option « *rechercher les mots dans l'ordre exact* »

Les objets les plus fréquemment utilisés dans Word et Excel sont présentés en annexes.

5.5.2. Accès aux objets

Pour accéder à un objet, il faut indiquer le chemin à parcourir pour l'atteindre dans la hiérarchie des objets. A partir de l'objet "racine" (*Application*), on applique en cascade des méthodes ou des propriétés pour parvenir à cet objet. Une fois celui-ci atteint, on peut lui appliquer une méthode ou propriété comme expliqué au 5.2 :

Application.ActiveWindow.WindowState = wdWindowStateMaximize

Ici, la propriété *ActiveWindow* de l'*Application* renvoie un objet *Window* (qui est la fenêtre active). La constante d'agrandissement (*wdWindowStateMaximize*) est affectée à sa propriété *WindowState*.

Si l'on doit accéder à plusieurs propriété d'un objet, ou lui appliquer plusieurs méthodes, le mécanisme précédent peut devenir lourd. Aussi est-il possible, pour alléger le code, d'utiliser l'instruction *With <un objet> … End With*, qui permet de définir plusieurs propriétés de cet objet en en évitant la réécriture. Par exemple,

```
With Application.Documents("VBA")
    .Activate                       ' Equivaut à Application.Documents("VBA.doc").Activate
    .EmbedTrueTypeFonts = True      '⇔ Application.Documents("VBA.doc").EmbedTrueTypeFonts = True
    With .Paragraphs(1)
        .Alignment = wdAlignParagraphJustify
        '⇔ Application.Documents("VBA.doc").Paragraphs(1).Alignment = wdAlignParagraphJustify
        nbCaracPremPara = .Range.Characters.Count
        '⇔ nbCaracPremPara = Application.Documents("VBA.doc").Paragraphs(1).Range.Characters.Count
    End With
End With
```

Concernant l'accès aux objets, voir encore les remarques [1] et [2] au bas de cette page.

5.5.3. Propriétés et méthodes globales

Afin d'alléger la notation, la plupart des propriétés et méthodes associées à l'objet *Application* peuvent être utilisées sans qualifier explicitement cet objet ; elles sont directement accessibles (par exemple, *ActiveDocument* ou *Selection* sous Word, *ActiveWindow* ou *ActiveCell* sous Excel n'ont pas besoin d'être précédés de « *Application.* » pour être utilisées). Les propriétés et méthodes qui peuvent être utilisées sans le qualificateur *Application* sont considérées comme « ***globales*** ». Pour afficher celles-ci, dans l'Explorateur d'objets, cliquez sur « *global* » en haut de la liste figurant dans la zone *Classes*. Dans cette liste figurent aussi les constantes globales prédéfinies (*cf.* cette notion plus loin).

---

[1] Les méthodes et propriétés peuvent être appliquées en cascade, par exemple :

Documents.Add.Save

Ici, la propriété *Documents* (sous-entendu, de *Application*) retourne la collection *Documents*, à laquelle la méthode *Add* ajoute un objet *Document* en retournant cet objet. La méthode *Save* est ensuite appliquée à cet objet *Document.*

[2] Pour connaître la hiérarchie d'un objet, et les propriétés et méthodes qui lui sont applicables, rendez-vous dans l'***Explorateur d'objets*** (en appuyant sur F2). Par exemple, dans Word, la méthode *Close* de l'objet *Document* indique :

Sub Close([SaveChanges], [OriginalFormat], [RouteDocument])
Membre de Word.Document

Les 3 paramètres optionnels de cette méthode sont affichés (entre crochets), de même que la hiérarchie de l'objet sur lequel elle s'applique. Les liens hypertextes permettent de naviguer dans cette hiérarchie.

6 – Structure du langage VBA

6.1. Types de données

Comme on a pu s'en apercevoir jusqu'ici, certaines propriétés d'objets contiennent du texte ; d'autres n'acceptent que des valeurs numériques ou des booléens. De même, lorsque vous déclarez une variable, vous pouvez préciser, en plus de son nom, le type de données qu'elle doit contenir. Cette section présente les types de données reconnus par VBA, valables pour les propriétés des objets comme pour les variables définies par l'utilisateur.

6.1.1. Types standards

Ces types sont communs à la plupart des langages de programmation (cf. tableau ci-après).

Type	Description	Notes	Taille (octets)
Byte	Octet	Plage : 0..255	1
Boolean	Booléen	Ne peut prendre que les valeurs True ou False	2
Integer	Entier	Plage : -32 768..32 767	2
Long	Entier long	Plage : environ $-2.10^9..+2.10^9$	4
Single	Réel en simple précision	Plages : $-3,4.10^{38}..-1,4.10^{-45}$ et $1,4.10^{-45}..3,4.10^{38}$	4
Double	Réel en double précision	Plages : $-1,8.10^{308}..-4,9.10^{-324}$ et $4,9.10^{-324}..1,8.10^{308}$	8
Currency	Nombre à virgule fixe (ex. monétaire)	Plage : environ $–922.10^9..+922.10^9$	8
Date	Date	Plages : 01/01/0100..31/12/9999 et 00:00:00..23:59:59	8
String (lg variable)	Chaîne de longueur variable	0 à environ 2 milliards de caractères	10+longueur de la chaîne
String (lg fixe)	Chaîne de longueur fixe	0 à environ 65 400 caractères	longueur de la chaîne

6.1.2. Types particuliers

Visual Basic possède en outre trois types particuliers de variables :

- les types de données ***définis par l'utilisateur*** s'apparentent aux *enregistrements* des langages C et Pascal ; ils sont créés grâce à l'instruction *Type* au niveau module :

```
Type NomTypePerso
        NomElément1 As Type
        NomElément2 As Type
        …
End Type
```

  Ce type s'utilise alors comme n'importe quel autre ; pour accéder aux éléments, employer la notation pointée.

- les variables de type *Variant* peuvent contenir des données de tout type (sauf une chaîne de longueur fixe et un type défini par l'utilisateur)[1]. Ceci simplifie l'écriture et offre plus de souplesse dans le traitement des données, car les conversions d'un type à l'autre sont alors automatiques et transparentes. Par exemple,

[1] Les variables de type *Variant* peuvent aussi contenir les valeurs spéciales suivantes :
- *Null* indique que la variable ne contient aucune donnée valide ;
- *Empty* désigne une variable non initialisée (la variable de type *Variant* équivaut alors à 0 si elle est utilisée dans un contexte numérique et à une chaîne de longueur nulle ("") dans un contexte de chaîne) ;
- *Error* permet d'indiquer qu'une condition d'erreur s'est produite dans une procédure.

une variable de ce type pourra, selon le contexte et les nécessités de l'exécution, prendre la valeur numérique 13 ou être représentée sous forme de la chaîne de caractères "13" ; il sera possible d'appliquer à cette variable des opérateurs ou fonctions portant aussi bien sur les nombres (ex. + ou *cos*) que sur les chaînes (ex. *&* ou *Len*).

Attention : il faut quand même savoir que le type *Variant* est gourmand en mémoire.

- les variables destinées à contenir des objets peuvent prendre comme type soit l'un de ceux prédéfinis dans l'application hôte (ex. *Range*), soit le type *Object*. En réalité, ces variables se déclarent comme n'importe quelle autre (voir paragraphe suivant), mais pour leur affecter une valeur, il faut utiliser l'instruction *Set*[1].

6.2. Variables et constantes

6.2.1. Déclaration

Il existe deux façons de déclarer des variables : implicitement ou explicitement. En effet, la déclaration des variables est facultative : l'apparition d'un mot non reconnu dans une affectation crée implicitement une nouvelle variable de type *Variant*. La déclaration explicite de variables se fait généralement par l'instruction *Dim*, parfois par *Public*, *Private* ou *Static* si l'on souhaite modifier la portée ou la durée de vie de la variable (voir plus loin). Dans tous les cas, la syntaxe (simplifiée) est la suivante :

Dim nomVar *[* **(** indices **)** *]* *[***As*** type*]*

Il est possible de déclarer plusieurs variables avec une seule instruction, en séparant par une virgule les différents noms et en spécifiant le type pour chacune. Par ailleurs, le type d'une variable peut être spécifié de manière implicite en suffixant son nom par l'un des caractères du tableau suivant.

Suffixe	Type de données	Suffixe	Type de données
%	Integer	#	Double
&	Long	@	Currency
!	Single	$	String

Le tableau ci-dessous donne des exemples de déclarations de variables.

Instruction	Déclaration de…	Exemples
Dim *nomVar*	Variable de type par défaut (*Variant*)	**Dim** anyValue
Dim *nomVarSuffixé*	Variable de type donné par le suffixe	**Dim** Nom$
Dim *nomVar* **As** *type*	Variable de type donné explicitement	**Dim** x **As** Integer
Dim *nomVar* **As** *String* ***** *taille*	Variable chaîne de taille fixe donnée	**Dim** x **As** String ***** 5
Dim *premVar, secVar* **As** *type*	Deux variables en une seule instruction	**Dim** unVariant, choix **As** Boolean
Dim *nomVar***(***taille***) As** *type*	Tableau unidimensionnel de type donné[2]	**Dim** tabJours(50) **As** Date
Dim *nomVar***(***taille***,***taille***)**	Tableau bidimensionnel (de *Variant*)	**Dim** matrice(3,4) As Integer
Dim *nomVar***(***indice* **To** *indice***)**	Tableau avec limites d'indices explicites	**Dim** mat(1 To 5, 4 To 9) As Double
Dim *nomVar***()As** *type*	Tableau dynamique (redimensionnable[3])	**Dim** MyArray() **As** Single

[1] En réalité, une variable objet ne contient qu'une ***référence*** à un objet. Lorsque vous utilisez l'instruction *Set*, la variable « pointe » désormais vers un objet, mais aucune copie de ce dernier n'est créée (pour créer réellement une nouvelle instance de l'objet, utilisez le mot clé *New* dans l'instruction *Set*). Si plusieurs variables objets font référence au même objet, toute modification apportée à l'objet est répercutée sur toutes les variables associées. Par ailleurs, pour mettre fin à l'association entre la variable et l'objet, utilisez la syntaxe : ***Set*** NomVariable = ***Nothing***.

[2] Les indices des tableaux commencent par défaut à 0, à moins d'avoir spécifié *Option Base 1*. La déclaration *Dim tabJours(50)* crée donc un tableau de 51 éléments (indices de 0 à 50).

[3] L'instruction *ReDim* permet de changer le nombre d'éléments, les dimensions, ou les limites inférieure et supérieure de chaque dimension. Par défaut, les valeurs sont perdues lors du redimensionnement, sauf si l'on utilise *Redim Preserve*.

Une ***constante*** permet d'associer un nom à une valeur : ce sont des chaînes de caractères, mais en réalité, elles représentent en général des valeurs numériques. Par exemple, à chacune des touches correspond une constante (*vbKeyTab*, *vbKeyA*, …), ce qui évite d'avoir à se souvenir de son code ASCII. Les constantes améliorent la lisibilité du code et rendent l'application plus facile à maintenir.

Lorsqu'une propriété accepte un nombre déterminé d'états, ceux-ci correspondent souvent à des constantes (ex. pour définir l'état d'une fenêtre, la propriété *WindowState* d'un objet *Window* peut prendre les valeurs *wdWindowStateMaximize*, *wdWindowStateMinimize* ou *wdWindowStateNormal*). VBA comporte plusieurs dizaines de constantes prédéfinies qui conservent la même valeur et sont accessibles de n'importe où dans le code. Les constantes utilisées par les objets Word commencent par les lettres *wd*, dans Excel par *xl*, dans Powerpoint par *pp*, celles communes avec Visual Basic par *vb*, avec Microsoft Office par *mso*. Il est possible de déclarer ses propre constantes grâce à la syntaxe :

Const nomConst *[**As** type]* **=** expression

où *Type* ne peut être ni un type personnalisé, ni *Object*.

6.2.2. Portée

La ***visibilité*** (ou ***accessibilité***) est la disponibilité d'une variable, constante ou procédure, dans le reste du code. La portion de code dans laquelle une variable est visible est appelée ***portée***. En VBA, il existe 3 niveaux de portée : niveau de procédure, niveau de module privé et niveau de module public. La déclaration d'une variable se fait donc soit dans une procédure (*Function*, *Property* ou *Sub*), soit au niveau d'un module (dans la section des déclarations, avant les procédures). Généralement, la déclaration s'effectue avec l'instruction *Dim*, mais les mots-clés *Public* et *Private* permettent de nuancer la portée des variables déclarées, comme l'indique le tableau suivant :

Instruction	Déclaration au niveau…	Accessibilité de la variable/constante ainsi déclarée
Dim	Procédure	Locale à la procédure
	Module	Locale au module (accessible de toutes les procédures du module)
Public	Procédure	<*Interdit* : le mot-clé *Public* est réservé au niveau module>
	Module	Toutes les procédures de tous les modules du projet
Private	Procédure	Locale à la procédure (comme avec *Dim*)
	Module	Locale au module (comme avec *Dim*)

Remarque : La déclaration des procédures peut aussi être précédée des mots-clés *Public* (valeur par défaut) ou *Private*. Dans ce dernier cas, la procédure n'est accessible qu'aux procédures du même module ; de plus, elle ne peut plus être affectée à un menu, une touche de raccourci ou un contrôle, ni être exécutée comme une macro indépendante.

6.2.3. Durée de vie

Dans une procédure, toute variable est (ré)initialisée lors de sa déclaration (nombres à *0*, chaînes à *""* (chaîne vide), variables de type *variant* à *Empty*, objets à *Nothing*). L'instruction *Static* permet, au niveau procédure, de déclarer des variables conservant leur valeur pendant toute la durée de l'exécution du module. La ***durée de vie*** de ces variables est alors identique à celle d'une variable de niveau module. Il est aussi possible de déclarer une procédure comme *Static*, moyennant quoi toutes les variables déclarées dans cette procédure seront statiques.

6.3. Instructions

6.3.1. Différents types d'instructions

- Les instructions de *déclaration* servent à nommer une variable, constante ou procédure[1]

  Exemples : *Sub uneProcedure()* , *Dim uneVariable As String* , *Const uneConstante="Salut"*

- Les instructions d'*affectation* utilisent l'opérateur = pour affecter une valeur à une variable, constante ou propriété. Ainsi, pour stocker dans une variable la valeur d'une propriété d'un objet, ou affecter une valeur à une propriété, on utilise respectivement les syntaxes :

  nomVar = Expression.Propriété ou Expression.Propriété = valeur

  où *Expression* a pour résultat un objet (la variable et la propriété doivent avoir le même type).

- Les instructions *exécutables* accomplissent des actions en appelant une méthode ou une fonction. Les instructions de contrôle (cf. paragraphe suivant) sont aussi considérées comme exécutables. L'exécution d'une méthode sur un objet se fait suivant la syntaxe déjà rencontrée *Objet.méthode*.

  Les ***commentaires*** sont des instructions particulières introduites par le mot-clé *Rem* ou simplement une apostrophe (') ; tout ce qui suit sur la même ligne est ignoré à l'exécution.

6.3.2. Structures de contrôle

Les structures de contrôles servent à effectuer des boucles ou des branchements conditionnels ; on les retrouve dans tous les langages de programmation structurés. Le tableau ci-dessous résume celles dont on dispose en VBA.

Type	Syntaxe(s)	Effet
Boucles	***Do Until*** *condition* *instructions* ***Loop***	Exécute les *instructions* jusqu'à ce que la *condition* soit vraie. Si la *condition* est vraie dès le premier test, les *instructions* ne sont jamais exécutées.
	Do While *condition* *instructions* ***Loop***	Exécute les *instructions* tant que la *condition* est vraie. Si la *condition* est fausse dès le premier test, les *instructions* ne sont jamais exécutées.
	Do *instructions* ***Loop Until*** *condition*	Exécute les *instructions* jusqu'à ce que la *condition* soit vraie. Les *instructions* sont au moins exécutées une fois.
	Do *instructions* ***Loop While*** *condition*	Exécute les *instructions* tant que la *condition* est vraie. Les *instructions* sont au moins exécutées une fois.
	For *compteur=début* ***to*** *fin [****step*** *pas]* *Instructions* ***Next*** *compteur*	Exécute les *instructions* le nombre de fois déterminé par le *compteur*. Ce dernier compte de *début* à *fin* par *pas* spécifié (*pas* de 1 par défaut).
	For Each *element* ***In*** *groupe* *instructions* ***Next*** *element*	Pour chaque élément du *groupe* spécifié (tableau ou collection), exécute les *instructions* en utilisant le nom *element* pour désigner le n-ième élément.

[1] Les noms de procédures, constantes, variables et arguments suivent les restrictions suivantes :
- le premier caractère doit être une lettre ;
- majuscules et minuscules ne sont pas différenciées, bien que la casse soit respectée ;
- ne pas utiliser de point, d'espace, ni $, !, #, & et @. En règle générale, n'utiliser que des lettres (éventuellement accentuées), des chiffres et le trait de soulignement (_) ;
- ne pas utiliser les noms réservés de Visual Basic (rechercher *mots clés* dans l'aide pour une liste complète) ;
- ne pas déclarer plusieurs fois le même nom de variable ou de constante dans un même niveau de portée ;
- ne pas dépasser 255 caractères ni donner à une *Function* un nom ressemblant à une adresse de cellule Excel !…

Instructions conditionnelles	***If*** *condition* ***Then*** *instructions* *(Le tout sur une seule ligne)*	Exécute les *instructions* si la *condition* est vraie.
	If *condition* ***Then*** *instructions1* *[**Else*** *instructions2]* ***End If***	Exécute les *instructions1* si la *condition* est vraie. Si la clause ***Else*** est présente, alors exécute les *instructions2* si la *condition* n'est pas vraie.
	If *condition1* ***Then*** *instructions1* ***ElseIf*** *condition2* ***Then*** *instructions2* ***ElseIf*** *condition3* ***Then*** *instructions3* *…* *[**Else*** *instructionsElse]* ***End If***	Exécute les *instructions1* si la *condition1* est vraie ; dans le cas contraire, fait un nouveau test sur *condition2*. Si celle-ci est vraie, les *instructions2* sont exécutées ; sinon, on fait un nouveau test sur *condition3*, et ainsi de suite. Si la clause ***Else*** est présente, alors les *instructionsElse* sont exécutées si toutes les conditions précédentes sont fausses.
	Select Case *expressionTestée* ***Case*** *listeValeurs1* *instructions1* ***Case*** *listeValeurs2* *instructions2* *…* *[**Case Else*** *instructionsElse]* ***End Select***	En fonction de la valeur d'une *expressionTestée*, branche le programme sur tel ou tel jeu d'instructions. Si l'expression a pour valeur l'une de celles contenues dans *listeValeurs1*, ce sont les *instructions1* qui sont exécutées, et ainsi de suite pour toutes les listes de valeurs. Si la clause *Case else* est présente et qu'aucune liste de valeurs ne contient l'*expressionTestée*, alors ce sont les *instructionsElse* qui sont exécutées en définitive.

L'instruction *Exit For* permet de quitter une boucle *For* ou *For Each* (l'exécution se continue à la ligne suivant le *Next*), et *Exit Do* quitte directement une boucle *Do*. Ces instructions sont toutefois à éviter, dans la mesure du possible, car elles nuisent à la lisibilité du code.

6.3.3. Gestion des erreurs

Pour éviter l'interruption de l'exécution lorsqu'une erreur survient, il est possible d'indiquer, dans le programme même, la marche à suivre dans ce cas grâce aux instructions ci-dessous. De manière générale, un bon programmeur essaie de prévoir le maximum possible de cas d'erreurs et de les traiter, car certaines erreurs n'interrompent pas le déroulement de l'exécution mais peuvent néanmoins entraîner un comportement imprévisible.

Le traitement des erreurs comporte les instructions suivantes :

- *On Error …*, qui indique ce que doit faire le programme lorsqu'une erreur est générée ; 3 solutions sont possibles :
  - *On Error Goto* ligne branche le programme vers une routine de gestion d'erreur nommée *ligne* ;
  - *On Error Resume Next* spécifie que l'exécution doit se poursuivre à la ligne suivante ;
  - *On Error Goto 0* désactive la gestion des erreurs (annule un précédent *On Error…*)
- *Resume …* , placée à la fin de la routine de gestion d'erreur, qui indique où continuer l'exécution une fois l'erreur traitée. Là encore, 3 possibilités s'offrent à nous :
  - *Resume 0* reprend l'exécution à la ligne même qui a généré l'erreur ;
  - *Resume Next* reprend l'exécution à la ligne qui suit celle qui a généré l'erreur ;
  - *Resume* etiquette reprend l'exécution à la ligne portant l'étiquette spécifiée.

Signalons encore que l'objet *Err* contient des informations relatives aux erreurs survenant lors de l'exécution d'un code, et que deux autres instructions peuvent être utiles dans ce contexte :

- *Error (*codeErreur*)* retourne le message associé à l'erreur portant le code spécifié ;
- *Error* codeErreur permet de simuler l'occurrence de l'erreur de code spécifié.

Le schéma général de l'instruction *On Error Goto* ligne, pris sur l'exemple d'une procédure *Sub*, est le suivant (ils serait identique avec une *Function* ou une *Property*) :

```
Sub nomProcédure (…)
On Error Goto ligne
    …                ' corps de la procédure
Exit Sub             ' évite d'exécuter la routine de gestion d'erreur en l'absence d'erreur
ligne :              ' début de la routine de gestion d'erreur
    instructions si erreur
    Resume { 0 | Next | etiquette }  ' etiquette doit se trouver dans le corps de la même procédure
End Sub
```

7. Exécution pas à pas et débogage

Le menu *Débogage* de l'éditeur donne accès à plusieurs possibilités pour mettre au point le code saisi. Il est possible de contrôler l'exécution en la stoppant à tout moment et d'inspecter la valeur de variables ou d'expressions.

7.1. Contrôle de l'exécution

Lorsque vous exécutez une macro en ***mode pas à pas***, l'interpréteur de commandes s'arrête à chaque ligne de code. Lorsqu'il rencontre un appel à une procédure ou fonction, il peut :

- entrer dans le code de celle-ci, et ce de manière récursive (touche F8, mode *pas à pas détaillé*) ;
- ou bien exécuter l'appel comme une seule instruction, sans entrer dans le code de la procédure (touches Shift+F8, mode *pas à pas principal*).

Les ***points d'arrêt*** permettent de stopper l'exécution en un endroit donné du code, pour ensuite continuer en mode pas à pas ou inspecter la valeur d'une variable ou d'une expression. Pour placer un point d'arrêt au niveau d'une ligne de code, placez-y le curseur et appuyez sur F9 (un nouvel appui sur F9 annule le point d'arrêt) ; le menu *Débogage* permet alors d'exécuter le code jusqu'à ce point.

La réinitialisation du code (Menu *Exécution* ou bouton ) stoppe définitivement l'exécution (contrairement à  , qui marque une pause) et effectue une remise à zéro des variables. Ceci est utile lorsqu'une erreur est survenue : avant de relancer une exécution, une réinitialisation est obligatoire.

7.2. Inspection de variables

Il est à tout moment possible de connaître la valeur d'une variable ou d'une expression. Plusieurs possibilités se présentent pour cela :

- inclure dans le code une instruction *Debug.Print* nomVariable, ce qui a pour effet d'imprimer le contenu d'une variable dans l'objet *Debug*, c'est-à-dire la fenêtre d'exécution (si celle-ci est masquée, rendez-vous dans le menu *Affichage*) ;
- amener le curseur de la souris sur un nom de variable, de constante, ou une expression, et l'y laisser un petit moment, jusqu'à ce que la valeur correspondante soit affichée dans une bulle d'aide ;
- « espionner » la valeur en sélectionnant l'expression souhaitée, puis en sélectionnant le menu *Débogage/Espion express* (pour une inspection ponctuelle) ou *Débogage/Ajouter un espion* . Cette dernière option affiche en permanence la valeur de l'expression dans la fenêtre *Espions*, fenêtre qui permet de plus d'inspecter les objets en profondeur, en parcourant leur hiérarchie.