

**INTRODUCTION A LA PROGRAMMATION**

**AVEC**

**VISUAL BASIC POUR APPLICATION  
D'EXCEL**

# Visual Basic pour Application d'Excel

## Introduction

Visual Basic pour Applications (VBA) est le langage de programmation des applications de Microsoft Office (Ms Word, Excel, Ms Access ,...etc.). VBA permet d'automatiser les tâches et de créer des applications complètes.

VBA utilise le même langage que Microsoft Visual Basic (VB). Sauf que VB est un langage complet qui permet de développer des applications indépendantes et librement distribuables alors qu'une application réalisée en VBA est complètement liée au logiciel sous lequel elle a été créée (une application VBA créée sous Excel ne pourra pas se lancer sur un poste si Excel n'est pas installé).

VBA est un langage puissant, souple et facile à utiliser, et permet de réaliser très rapidement des applications performantes

## L'enregistreur de macros

Les applications MS Office Excel, Word , Access, ...etc possèdent un outil très pratique : l'enregistreur de macros. Cet outil transforme en langage VBA toutes les commandes effectuées par l'utilisateur dans l'application hôte. Il permet d'automatiser sans aucune connaissance de la programmation certaines de vos tâches et également de se familiariser avec le langage VBA.

### Utilisation de l'enregistreur de macros

Prenons un exemple :

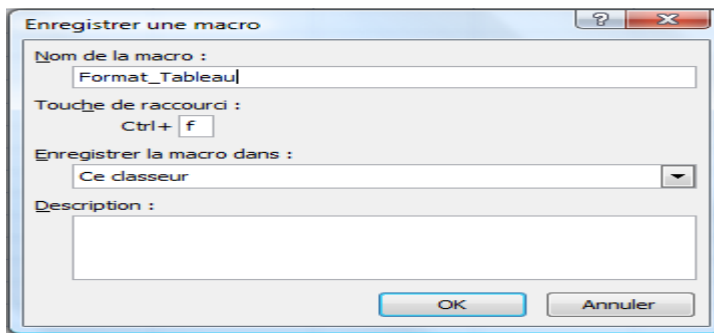
Supposons que vous voulez mettre au même format tous les tableaux que vous créez. Plutôt que de reproduire à chaque fois les mêmes commandes, vous allez utiliser l'enregistreur de macros. Pour ce faire procédez comme suit :

- 1- sélectionnez d'abord votre tableau. En effet, si vous le sélectionnez après avoir lancé l'enregistrement, la macro reproduirait cette sélection à chaque lancement et s'exécuterait toujours sur les mêmes cellules.

	A	B	C	D	E	F	G
1							
2			9H	10H	11H	12H	
3		LUNDI					
4		MARDI					
5		MERCREDI					
6		JEUDI					
7		VENDREDI					
8		SAMEDI					
9		DIMANCHE					
10							

- 2- Lancez l'enregistrement par la commande :

Onglet Developpeur / Groupe Code /Outil Enregistrer une macro



**Nom de la macro :** Il est important de renommer de façon explicite la macro. Le nom de la macro doit commencer par une lettre et ne doit pas contenir d'espaces. Utilisez le caractère de soulignement pour séparer les mots.


**Touche de raccourci :** Il est possible de créer un raccourci clavier pour lancer la macro en saisissant une lettre. Vous pouvez utiliser les majuscules et lancer la macro par les touches Ctrl+MAJ+Lettre.

**Enregistrer la macro dans :** Classeur contenant la macro. La macro ne pourra ensuite s'exécuter que si le classeur dans lequel la macro a été enregistrée est ouvert. Si vous choisissez "classeur de macros personnelles", un nouveau classeur est créé et enregistré dans le dossier "xlstart" de façon que vos macros soient disponibles à chaque fois que vous utilisez Excel.

**Description :** Vous pouvez donner une description à votre macro.

- 3- Cliquez sur "OK", la macro va enregistrer toutes les commandes que vous allez utiliser. Ici, vous allez changer le format de votre tableau (Couleur, Polices, Encadrement).

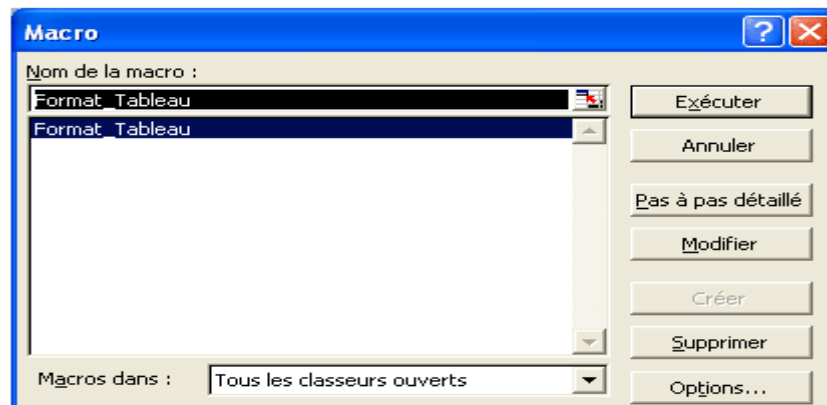
	A	B	C	D	E	F	G
1							
2			9H	10H	11H	12H	
3		LUNDI					
4		MARDI					
5		MERCREDI					
6		JEUDI					
7		VENDREDI					
8		SAMEDI					
9		DIMANCHE					
10							

- 4- arrêter ensuite l'enregistrement en cliquant sur le bouton  qui se trouve dans la barre d'état ou sur l'onglet Developpeur/Groupe Code/Outil Arrêter l'enregistrement.

- Pour reproduire le même format sur un nouveau du tableau :

	A	B	C	D	E	F	G	H	I	J	K	L	M
11													
12													
13			1	2	3	4	5	6	9	10	11	12	
14		JANVIER											
15		FEVRIER											
16		MARS											
17		AVRIL											
18		MAI											
19		JUIN											
20		JUILLET											
21		AOÛT											
22		SEPTEMBRE											
23		OCTOBRE											
24		NOVEMBRE											
25		DÉCEMBRE											
26													

- Sélectionnez le nouveau tableau et utilisez la macro précédemment créée (à condition que le classeur dans laquelle elle a été enregistrée soit ouvert).
- Pour la lancer Cliquez sur l'onglet Developpeur/Groupe Code/Outil Macros



- Sélectionner la macro désirée puis cliquez sur "Exécuter" :

	A	B	C	D	E	F	G	H	I	J	K	L	M
11													
12													
13			1	2	3	4	5	6	9	10	11	12	
14		JANVIER											
15		FEVRIER											
16		MARS											
17		AVRIL											
18		MAI											
19		JUIN											
20		JUILLET											
21		AOÛT											
22		SEPTEMBRE											
23		OCTOBRE											
24		NOVEMBRE											
25		DÉCEMBRE											
26													

- Vous auriez également pu lancer la macro par le raccourci clavier "Ctrl+f" défini lors de sa création.

Quelques précautions à prendre lors de son utilisation :

- Bien penser aux commandes que l'on veut enregistrer pour éviter de créer du code inutile.
- Savoir dans quel classeur enregistrer la macro pour son utilisation future.
- Renommer les macros de façon explicite.

Limites de l'enregistreur de macros :

Il ne peut que traduire les commandes réalisées par l'utilisateur dans l'application hôte et écrit en général plus de lignes de code que nécessaire.

Lors de l'écriture de macros, nous verrons comment utiliser et réduire le code créé par l'enregistreur.

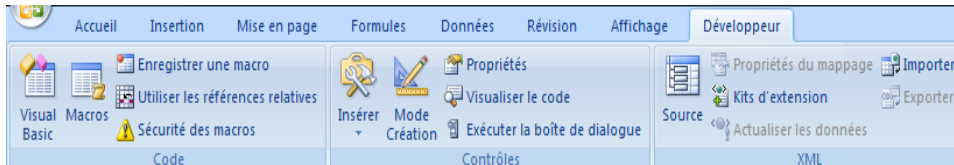
Les macros créées sont visibles et modifiables via l'éditeur de macro VBE (Visual Basic Editor).

# Environnement de programmation en VBA

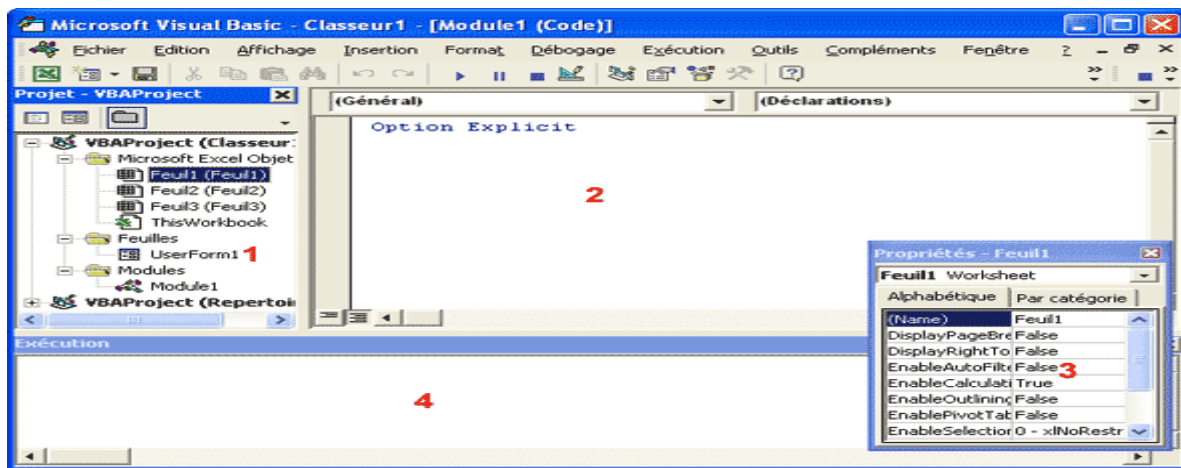
## Editeur Visual Basic (VBE)

VBE (Visual Basic Editor) est l'environnement de programmation de VBA. Il se lance par le raccourci clavier "Alt+F11" ou par la commande :

**Onglet Developpeur/Groupe Code / Outil Visual Basic**



### Les principales fenêtres de VBE :



1. **Fenêtre VBAProject.** Elle présente les différents projets ouverts et permet de naviguer facilement entre vos différentes feuilles de codes VBA.
2. **Fenêtre Code.** C'est l'endroit où vous allez saisir votre code VBA.
3. **Fenêtre Propriétés.** Propriétés de l'objet sélectionné.
4. **Fenêtre Exécution.** Elle permet de tester une partie du code. Elle peut s'avérer très utile pour voir comment s'exécutent certaines lignes de code.

Pour afficher ou masquer une fenêtre, exécuter la commande correspondante du menu Affichage .

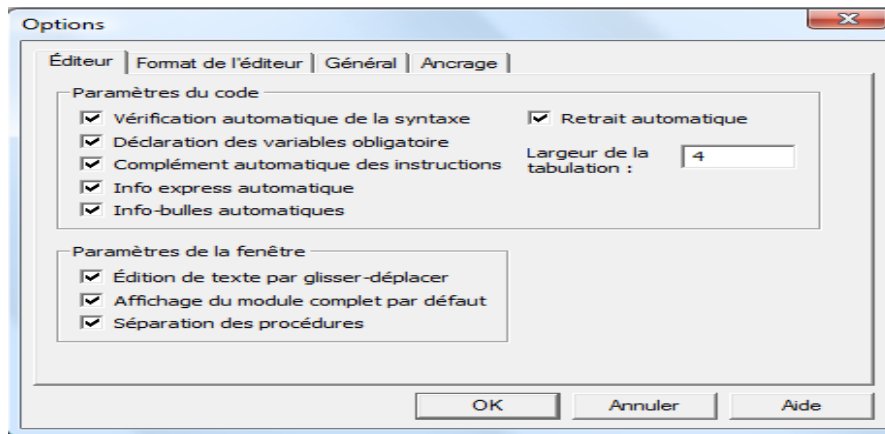
### Configurer l'éditeur VBE

Il est important de bien configurer l'éditeur VBE. En effet, VBE peut vous aider dans l'écriture de votre code et le mettre en forme de façon à ce qu'il soit plus facile à lire.

Sous VBE, lancer le menu "Outils-Options" :

### Onglet Editeur :

Larbi HASSOUNI



### Vérification automatique de la syntaxe :

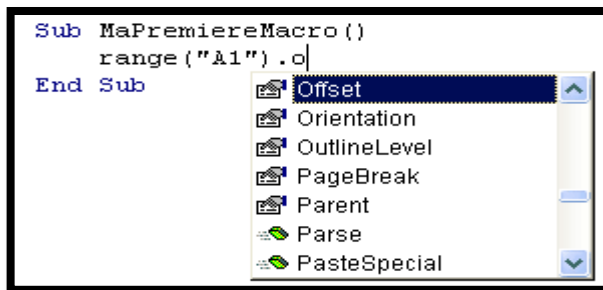
Vérification automatiquement de la syntaxe lors de la saisie d'une ligne de code.

### Déclarations de variables obligatoires :

Sous VB, la déclaration de variables n'est pas obligatoire. Cependant, il est recommandé de cocher cette option. Ce qui ajouterait automatiquement l'instruction "Option Explicit" dans la partie des déclarations générales de tout nouveau module, rendant ainsi les déclarations des variables obligatoire (Voir variables dans la suite du cours).

### Complément automatique des instructions :

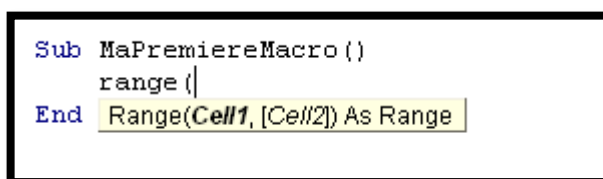
Cette option permet à VBE de vous aider dans la saisie de votre code :



Vous comprendrez très vite son utilité lorsque vous saisirez vos premières lignes de codes.

### Info express automatique :

Encore une option très utile. Elle affiche les différents arguments que possède la fonction que vous venez de taper :



### Info-bulles automatique :

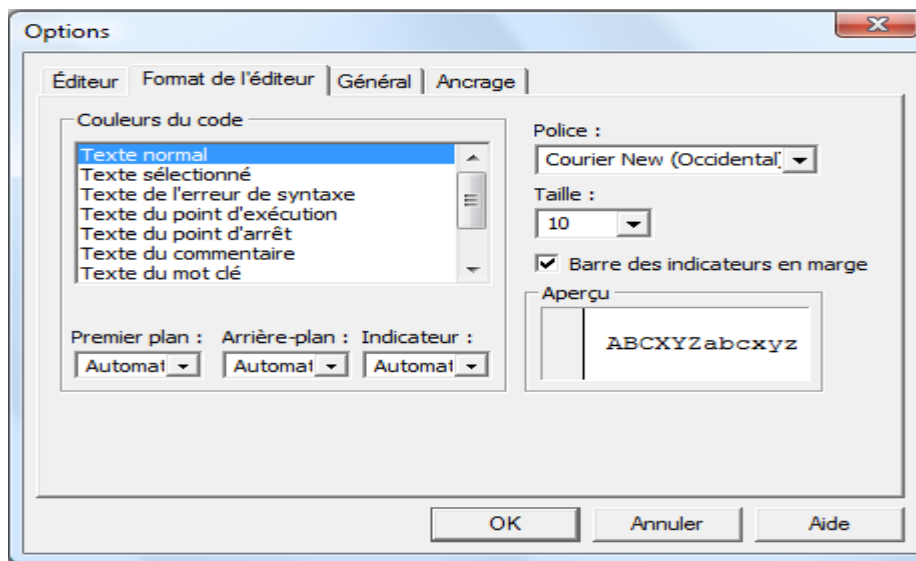
Indispensable lors d'un débogage pas à pas. Elle permet l'affichage de vos variables.

**Retrait automatique :**

Permet à VBE de placer chaque ligne de code au même niveau que la ligne précédente. Le retrait de lignes se fait par les touches "Tab" et "Shift+Tab". Cette option est nécessaire pour une bonne lecture du code VBA.

**Paramètres de la fenêtre :**

Les 3 options sont intéressantes. L'édition de texte par glisser-déplacer permet de déplacer à l'aide de la souris le bloc de code sélectionné, l'affichage du module complet par défaut permet l'affichage de toutes les procédures d'un même module et la séparation des procédures oblige VBE à créer des traits entre deux procédures.

**2 - Onglet Format de l'éditeur :**

Cet onglet permet de changer la police et son format pour les différentes parties du code inscrit dans vos modules.

Ces options vous permettent de configurer à votre convenance l'éditeur de macros. Si vous débutez, il est conseillé cependant de garder les options par défaut.



# Modèle Objet d'Excel

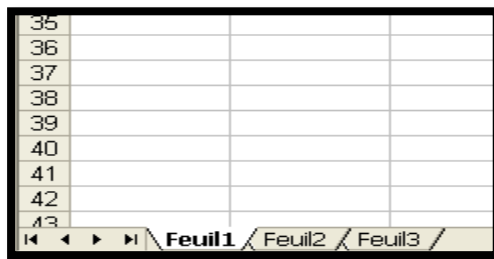
VBA manipule les objets de l'application hôte. Chaque objet possède des propriétés et des méthodes.

## Les objets :

Chaque objet représente un élément de l'application.

Sous Excel, un classeur, une feuille de calcul, une cellule, un bouton, etc ... sont des objets. Par exemple, l'objet Application représente l'application Excel, Workbook l'objet classeur, Worksheet l'objet feuille de calcul etc...

Tous les objets de même type forment une collection comme, par exemple, toutes les feuilles de calcul d'un classeur forment la collection Worksheets. Chaque élément est alors identifié par son nom ou par un index.



Pour faire référence à la Feuil2, on va utiliser Worksheets(2) ou Worksheets("Feuil2")

Chaque objet peut avoir ses propres objets. Par exemple, l'objet Application possède des classeurs qui possèdent des feuilles qui possèdent des cellules.

Pour faire référence à une cellule, on pourrait ainsi utiliser :

```
Application.Workbooks(1).Worksheets("Feuil2").Range("A1")
```

## Les propriétés :

Une propriété correspond à une particularité de l'objet. La valeur d'une cellule, sa couleur, sa taille, etc...sont des propriétés de l'objet Range. Les objets sont séparés de leurs propriétés par un point. On écrira ainsi Cellule.Propriété=valeur :

```
'Mettre la valeur 100 dans la cellule A1  
Range("A1").Value = 100
```

Une propriété peut également faire référence à un état de l'objet. Par exemple, si on veut masquer la feuille de calcul "Feuil2", on écrira :

```
Worksheets("Feuil2").Visible = False
```

## Les méthodes :

On peut considérer qu'une méthode est une opération que réalise un objet. Les méthodes peuvent être considérées comme des verbes tels que ouvrir, fermer, sélectionner, enregistrer, imprimer, effacer, etc... Les objets sont séparés de leurs méthodes par un point.

Par exemple, pour sélectionner la feuille de calcul nommé "Feuil2", on écrira :

```
Worksheets("Feuil2").Select
```

Lorsque l'on fait appel à plusieurs propriétés ou méthodes d'un même objet, on fera appel au bloc d'instruction :

**With** Objet  
    *Instructions*  
**End With.**

Cette instruction rend le code souvent plus facile à lire et plus rapide à exécuter.

```
'Mettre la valeur 100 dans la cellule A1, la police en gras et en  
'italique et copier la cellule.  
With Worksheets("Feuil2").Range("A1")  
    .Value = 100  
    .Font.Bold = True  
    .Font.Italic = True  
    .Copy  
End With
```

Ce vocabulaire peut paraître déroutant mais deviendra très rapidement familier lors de la création de vos premières applications.

### Les évènements :

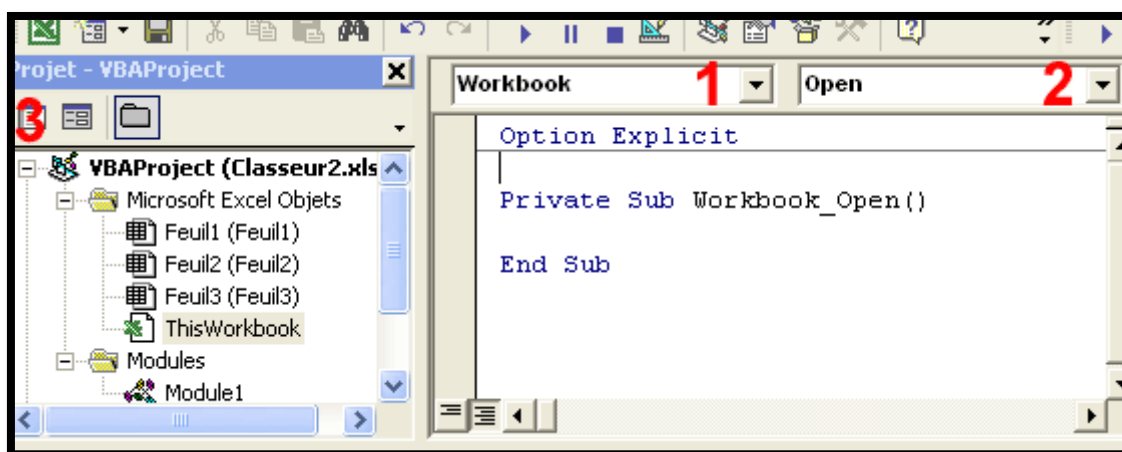
Pour qu'une macro se déclenche, il faut qu'un évènement (un clic sur un bouton, l'ouverture d'un classeur, etc...) se produise. Sans évènements, rien ne peut se produire.

### Les évènements liés aux objets.

Les principaux objets pouvant déclencher une macro sont :

- Un classeur
- Une feuille de travail
- Une boîte de dialogue

Chacun de ces objets possède leur propre module. Pour y accéder, lancer l'éditeur de macro :



Pour créer une procédure événementielle liée à un classeur, sélectionner le classeur "ThisWorkbook" puis cliquez sur l'icône 3 (ou plus simplement double-clic sur "ThisWorkbook").

Vous accédez ainsi au module lié à l'objet. Sélectionnez "Workbook" dans la liste 1 puis sur l'évènement désiré dans la liste 2.

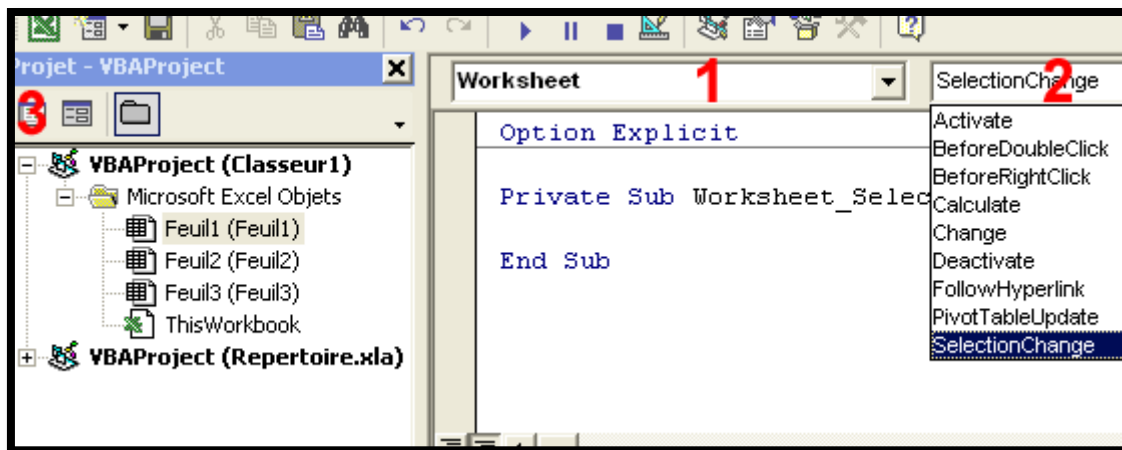
Par exemple, le code suivant lancera la procédure nommée "Test" à l'ouverture du classeur :

```
Private Sub Workbook_Open()  
Test  
End Sub
```

Liste des évènements de l'objet Workbook :

<i><b>Evénements:</b></i>	<i><b>Se produit :</b></i>
Activate	quand le classeur ou une feuille est activé
AddinInstall	quand le classeur est installé en macro complémentaire
AddinUninstall	quand le classeur est désinstallé en macro complémentaire
BeforeClose	avant que le classeur soit fermé
BeforePrint	avant l'impression du classeur
BeforeSave	avant l'enregistrement du classeur
Deactivate	quand le classeur ou une feuille est désactivé
NewSheet	lorsqu'une nouvelle feuille est créée
Open	à l'ouverture du classeur
PivotTableCloseConnection	lorsqu'un qu'un rapport de tableau croisé dynamique se déconnecte de sa source de données
PivotTableOpenConnection	lorsqu'un qu'un rapport de tableau croisé dynamique se connecte à une source de données
SheetActivate	lorsqu'une feuille est activée
SheetBeforeDoubleClick	lors d'un double-clic
SheetBeforeRightClick	lors d'un clic avec le bouton droit de la souris
SheetCalculate	après le recalcul d'une feuille de calcul
SheetChange	lors de la modification d'une cellule
SheetDeactivate	lorsqu'une feuille est désactivée
SheetFollowHyperlink	lors d'un clic sur un lien hypertexte
SheetPivotTableUpdate	lors de la mise à jour de la feuille du rapport de tableau croisé dynamique
SheetSelectionChange	lors d'un changement de sélection sur une feuille de calcul
WindowActivate	lorsqu'un classeur est activé
WindowDeactivate	lorsqu'un classeur est désactivé
WindowResize	lors du redimensionnement de la fenêtre d'un classeur

La création d'une procédure événementielle liée à une feuille de calcul se fait de la même façon.



Liste des évènements de l'objet Worksheet :

<i><b>Evénements:</b></i>	<i><b>Se produit :</b></i>
Activate	quand une feuille est activée
BeforeDoubleClick	lors d'un double-clic
BeforeRightClick	lors d'un clic avec le bouton droit de la souris
Calculate	après le recalcul de la feuille de calcul
Change	lors de la modification d'une cellule
Deactivate	quand une feuille est désactivée
FollowHyperlink	lors d'un clic sur un lien hypertexte
PivotTableUpdate	lorsqu'un rapport de tableau croisé dynamique a été mis à jour
SelectionChange	lors d'un changement de sélection

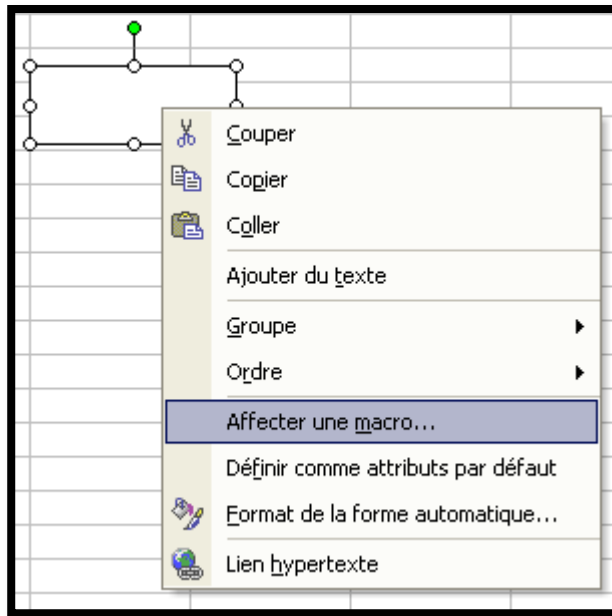
Certaines procédures événementielles possèdent des paramètres tels que "**Cancel**", qui peut annuler la procédure. Par exemple, pour empêcher l'impression du classeur, on utilisera :

```
Private Sub Workbook_BeforePrint(Cancel As Boolean)
    Cancel = True
End Sub
```

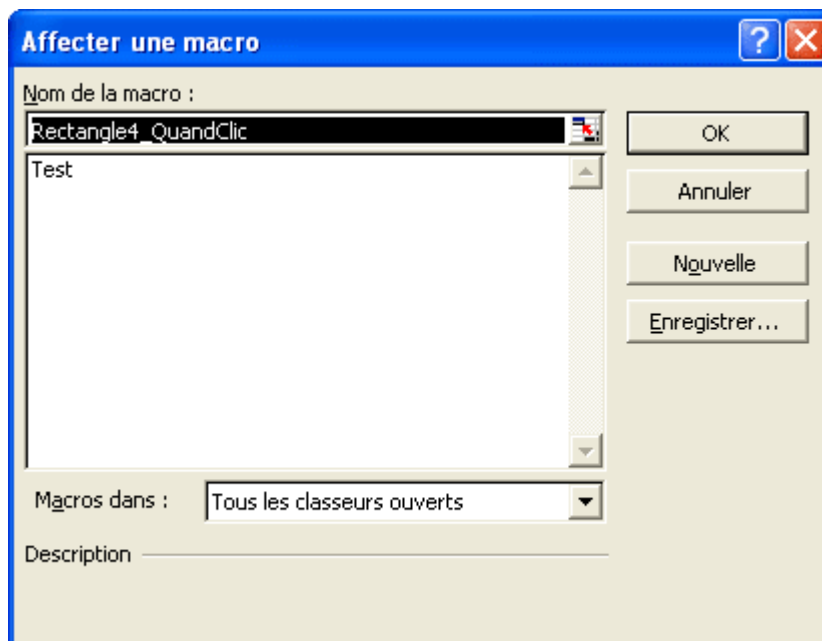
Pour récupérer la valeur d'une cellule modifiée, on utilisera :

```
Private Sub Worksheet_Change(ByVal Target As Range)
    MsgBox Target.Value
End Sub
```

Une macro peut également se déclencher en cliquant sur un élément graphique de l'application (Une image, une zone de texte, un objet WordArt, un rectangle ...). Créez un élément puis cliquez sur "Affecter une macro" dans le menu contextuel.



Cliquez sur le nom de la macro désirée puis validez.



Un simple clic sur l'objet lancera la macro.

Il existe également des procédures événementielles liées aux boîtes de dialogues (Voir le cours sur les UserForms).

## Boîtes de dialogue avec l'utilisateur

Pour communiquer avec l'utilisateur, VBA fournit deux boîtes de dialogues :

1. La boîte MsgBox qui permet d'afficher des messages (donc des résultats),
2. la boîte InputBox qui permet de saisir des données d'entrée.

### La Boîte MsgBox

La boîte MsgBox peut simplement afficher une information. La procédure qui la contient est alors stoppée tant que l'utilisateur n'a pas cliqué sur le bouton « OK ».

La syntaxe est :

#### MsgBox Message

Exemple :

```
MsgBox "Bonjour"
```







Le texte peut-être affiché sur plusieurs lignes en utilisant le code retour chariot chr(13) ou le code retour ligne chr(10).

```
MsgBox "Bonjour" & Chr(10) & "Il est " & Time
```

Vous pouvez ajouter une icône concernant le type de message à afficher.

Les types d'attribut icône :

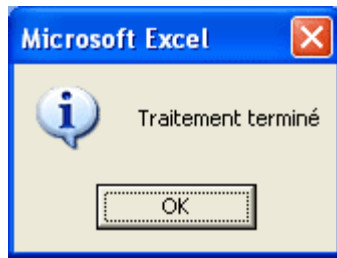
<i>Constante :</i>	<i>Icône :</i>	<i>Utilisation :</i>
vbCritical		Pour une erreur fatale
vbExclamation		Pour une remarque
vbInformation		Pour une information
vbQuestion		Pour une question

La syntaxe pour ajouter une icône est :

#### MsgBox "Message", attribut icône

Exemple :

```
MsgBox "Traitement terminé", vbInformation
```



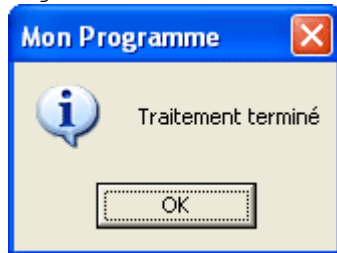
Le titre de la fenêtre (Microsoft Excel) peut être changé.

La syntaxe est :

**MsgBox "Message", attribut icône, "Titre de la fenêtre"**

Exemple :

`MsgBox "Traitement terminé", vbInformation, "Mon Programme"`



La boîte MsgBox peut également demander une information à l'utilisateur. Dans ce cas, la boîte de message comprend plusieurs boutons

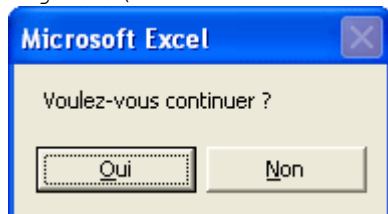
Les types d'attribut Boutons :

<i><b>Constante :</b></i>	<i><b>Boutons :</b></i>
vbAbortRetryIgnore	<input type="button" value="Abandonner"/> <input type="button" value="Recommencer"/> <input type="button" value="Ignorer"/>
vbOKCancel	<input type="button" value="OK"/> <input type="button" value="Annuler"/>
vbRetryCancel	<input type="button" value="Recommencer"/> <input type="button" value="Annuler"/>
vbYesNo	<input type="button" value="Oui"/> <input type="button" value="Non"/>
vbYesNoCancel	<input type="button" value="Oui"/> <input type="button" value="Non"/> <input type="button" value="Annuler"/>

La syntaxe est : **MsgBox ("Message", attribut bouton )**

Exemple :

`MsgBox ("Voulez-vous continuer ?", vbYesNo)`

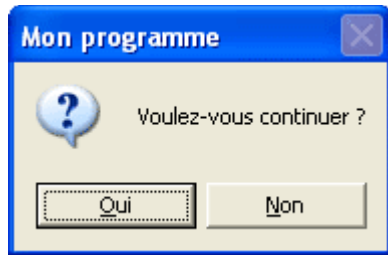


Vous pouvez également y ajouter les icônes et personnaliser le titre de la fenêtre en utilisant la syntaxe :

**Msgbox ("Message", attribut bouton + attribut icône, "titre de la fenêtre").**

Exemple :

```
MsgBox ("Voulez-vous continuer ?",vbYesNo+vbQuestion,"Mon programme")
```



MsgBox renvoie une valeur différente pour chaque bouton.

<i>Constante :</i>	<i>Valeur :</i>
vbOK	1
vbCancel	2
vbAbort	3
vbRetry	4
vbIgnore	5
vbYes	6
vbNo	7

Ainsi, si l'utilisateur clique sur le bouton "OK", MsgBox renvoie la valeur 1, sur le bouton "Annuler" la valeur 2, sur le bouton "Ignorer" la valeur 5 ...

Cette valeur peut être récupérée dans une variable.

```
'Dans la ligne d'instruction suivante, si l'utilisateur  
'clique sur le bouton "Oui", Reponse prendra comme valeur 6  
'sinon Reponse prendra comme valeur 7.
```

```
Reponse = MsgBox ("Voulez-vous continuer ?", vbYesNo)
```

```
'La ligne suivante arrête la procédure si l'utilisateur  
'clique sur "Non"
```

```
If Reponse = 7 Then Exit Sub
```



## La boîte InputBox

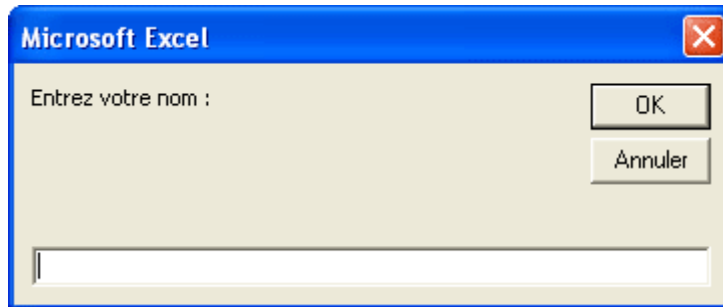
La boîte InputBox permet de demander à l'utilisateur d'entrer des données.

La syntaxe est :

**InputBox ("Message")**

Exemple :

```
InputBox ("Entrez votre nom :")
```



Comme pour la boîte MsgBox, vous pouvez changer le titre de la fenêtre. Vous pouvez également entrer une valeur par défaut dans la zone de saisie.

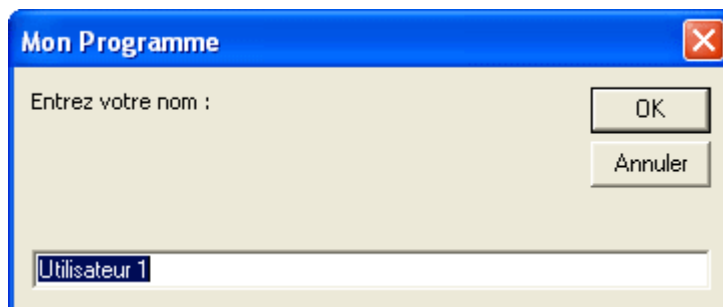
La syntaxe devient :

**InputBox ("Message", "Titre de la fenêtre", "Valeur par défaut").**

La valeur saisie peut être récupérée dans une variable. Si l'utilisateur clique sur le bouton "Annuler", la variable renvoie une chaîne de longueur nulle ("").

Exemple :

```
Message = InputBox("Entrez votre nom :", "Mon Programme", "Utilisateur 1")
```



```
Message = InputBox("Entrez votre nom :", "Mon Programme", "Utilisateur 1")
'La ligne suivante arrête la procédure si l'utilisateur
'clique sur "Annuler"
If Message = "" Then Exit Sub
'La ligne suivante place la valeur saisie dans la cellule
'A1 de la feuille active
Range("A1").Value = Message
```

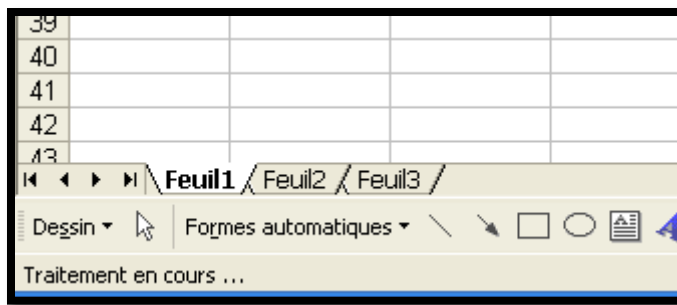
Vous pouvez également écrire un message dans la barre d'état de l'application

La syntaxe est :

**Application.StatusBar = "Message"**

Exemple :

```
Application.StatusBar = "Traitement en cours ..."
```



A la fin de la procédure, pensez à supprimer le message de la barre d'état par la ligne d'instruction: `Application.StatusBar = False`.

## Les variables

Une variable permet de stocker une donnée (valeur numérique, texte, valeur logique, date ...).

Elle peut également faire référence à un objet.

Suivant la nature des données que la variable recevra, on lui affectera un type différent.

Les types de données que fournit VBA et qu'on peut affecter à une variable de VB sont :

Type de données	Mot clé	Espace occupé	Plage de valeur
Octet	<b>Byte</b>	1 octet	Entier de 0 à 255
Logique	<b>Boolean</b>	2 octets	True ou False
Entier	<b>Integer</b>	2 octets	Entier de -32 768 à 32 768
Entier Long	<b>Long</b>	4 octets	Entier de -2 147 483 648 et 2 147 483 647 à 2 147 483 648 et 2 147 483 647
Décimal simple	<b>Single</b>	4 octets	-3,402823E38 à -1,401298E-45 pour les valeurs négatives 1,401298E-45 à 3,402823E38 pour les valeurs positives.
Décimal Double	<b>Double</b>	8 octets	-1,79769313486231E308 à -4,94065645841247E-324 pour les valeurs négatives 4,94065645841247E-324 et 1,79769313486231E308 pour les valeurs positives
Monétaire	<b>Currency</b>	8 octets	de -922 337 203 685 477,5808 et 922 337 203 685 477,5807
Date	<b>Date</b>	8 octets	1er Janvier 100 au 31 décembre 9999
Decimal	<b>Decimal</b>	12 octets	+/-79 228 162 514 264 337 593 543 950 335 sans point décimal +/-7,9228162514264337593543950335 avec 28 décimales.
Objet	<b>Object</b>	4 octets	toute référence à des objets
Chaîne de caractères à longueur variable	<b>String</b>	10 octets + longueur de chaîne	de 0 à 2 milliards de caractères
Chaîne de caractères à longueur fixe	<b>String</b>	Longueur de la chaîne	1 à 65 400 caractères
Variant avec chiffres	<b>Variant</b>	16 octets	Valeur numérique jusqu'au type double.
Variant avec caractères	<b>Variant</b>	22 octets + longueur de la chaîne	Même plage que pour un String de longueur variable
Défini par l'utilisateur	<b>Type</b>	Variable	Identique au type de données.

Pour rendre obligatoire la déclaration de variables, placez l'instruction "Option Explicit" sur la première ligne du module ou cochez l'option "Déclaration des variables obligatoires" dans le menu "Outils-Options" de l'éditeur VBE.

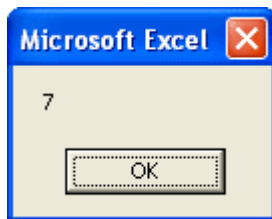
La déclaration explicite d'une variable se fait par le mot Dim (abréviation de Dimension). Le nombre maximum de caractères du nom de la variable est de 255. Il ne doit pas commencer par un chiffre et ne doit pas contenir d'espaces.

La syntaxe est :

**Dim NomDeLaVariable as Type**

**Exemple :**

```
Sub Test()  
    Dim SommeVal As Integer  
    Dim Val1 As Integer  
    Dim Val2 As Integer  
    Val1 = 5  
    Val2 = 2  
    SommeVal = Val1 + Val2  
    MsgBox Somme  
End Sub
```



Vous pouvez également déclarer vos variables sur une même ligne :

```
Sub Test()  
    Dim SommeVal As Integer, Val1 As Integer, Val2 As Integer  
    Val1 = 5  
    Val2 = 2  
    SommeVal = Val1 + Val2  
    MsgBox SommeVal  
End Sub
```

La portée d'une variable est différente suivant l'endroit et la façon dont elle est déclarée. Une variable déclarée à l'intérieur d'une procédure est dite "Locale". Elle peut-être déclarée par les mots Dim, Static ou Private. Dès que la procédure est terminée, la variable n'existe plus en mémoire sauf si elle est déclarée par le mot Static. Une variable Locale est généralement placée juste après la déclaration de la procédure.

```
Option Explicit  
'Les variables Val1 et Val2 sont libérées de la mémoire alors que  
la variable SommeVal garde sa valeur à la fin de la procédure  
Sub Test()  
    Static SommeVal As Integer  
    Dim As Val1, Integer, Val2 As Integer  
    'Instructions  
End Sub
```

Une variable peut être "Locale au module" si celle-ci est déclarée avant la première procédure d'un module. Toutes les procédures du module peuvent alors lui faire appel. Elle est déclarée par les mots Dim ou Private.

```

Option Explicit
'Les variables Val1 et Val2 peuvent être utilisées dans toutes les
procédures du module
Dim Val1 As Integer, Val2 As Integer

Sub Test()
    Static SommeVal As Integer
    SommeVal = Val1 + Val2
End Sub

Sub Test2()
    Static DivisVal As Integer
    DivisVal = Val1 / Val2
End Sub

```

Un variable peut également être accessible à tous les modules d'un projet. On dit alors qu'elle est publique. Elle est déclarée par le mot Public. Elle ne peut pas être déclarée dans un module de Feuille ou dans un module de UserForm.

```

Option Explicit
'Les variables Val1 et Val2 peuvent être utilisées dans toutes les
procédures de tous les modules du projet.
Public Val1 As Integer, Val2 As Integer

```

Une variable peut garder toujours la même valeur lors de l'exécution d'un programme. Dans ce cas, elle est déclarée par les mots Const ou Public Const.

```

Option Explicit
'La variable Chemin gardera toujours la valeur.
Const Chemin as String = "c:\application\excel\"

```

Il est possible de définir une taille fixe pour une variable de type String par la syntaxe :

**Dim Variable as String \* Longueur**

où Longueur correspond au nombre de caractère que prend la variable.

```

Option Explicit

Sub Test
    Dim Couleur as String * 5
    Couleur = "Rouge"
    ' Si Couleur était égal à "Orange", la variable Couleur aurait pris
    comme valeur "Orang".
End Sub

```

Il est important de déclarer ses variables par un nom explicite pour rendre le programme plus lisible. Vous pouvez également précéder ce nom par le caractère standard des types de variables. Par exemple, le caractère "i" représente un entier et la variable peut être nommée Dim iNombre as Integer.

<i>Caractère :</i>	<i>Type de variable :</i>
b	Boolean
i	Integer

l	long
s	Single
d	Double
c	Currency
dt	Date
obj	Object
str	String
v	Variant
u	Défini par l'utilisateur

Vous pouvez également créer vos propres types de données à l'intérieur du bloc "Type - End Type".

```
Option Explicit
'exemple de création d'un type de données personnalisé
Type Contacts
    Nom As String
    Prenom As String
    Age As Integer
End Type

Sub Test()
'Déclaration de la variable du type personnalisé
    Dim AjoutContact As Contacts
    AjoutContact.Nom = "TOTO"
    AjoutContact.Prenom = "Titi"
    AjoutContact.Age = 20
End Sub
```

Les variables peuvent également faire référence à des objets comme des cellules, des feuilles de calcul, des graphiques, des classeurs ... Elles sont déclarées de la même façon qu'une variable normale.

```
Option Explicit

Sub Test()
'La variable MaCel fait référence à une plage de cellule
    Dim MaCel As Range
'Le mot Set lui affecte la cellule "A1"
    Set MaCel = Range("A1")
'La cellule "A1" prend comme valeur 10
    MaCel.Value = 10
End Sub
```

**Classeurs**

**Feuilles**

**Plage de Cellules**

## Les classeurs.

Les classeurs sont désignés par le mot "Workbook". Ils peuvent être ouvert, fermé, enregistré, activé, masqué, supprimé ... par une instruction VB.

Quelques exemples d'instructions sur les classeurs :

```
'Ajouter un nouveau classeur  
Workbooks.Add
```

```
'Fermer un classeur. Le nom du classeur ou son index peut  
être indiqué.  
Workbooks("NomDuClasseur.xls").Close
```

```
'Fermer le classeur actif.  
ActiveWorkbook.Close
```

```
'Ouvrir un classeur.  
Workbooks.Open "c:\Chemin\NomDuFichier.xls"
```

```
'Activer un classeur.  
Workbooks("NomDuClasseur.xls").Activate
```

Certaines méthodes de l'objet Workbook possèdent des arguments.

Quelques exemples :

```
'Fermer un classeur sans l'enregistrer  
Workbooks("NomDuClasseur.xls").Close False
```

```
'Ouvrir un classeur en lecture seule.  
Workbooks.Open "c:\Chemin\NomDuFichier.xls", , True
```

```
'Enregistrer un classeur sous "Test.xls" avec comme mot de passe  
"testpass"  
Workbooks(1).SaveAs "test.xls", , "testpass"
```



## Les feuilles de calcul.

Les feuilles de calcul sont désignées par le mot "Worksheet". Comme les Workbook, ces objets possèdent de nombreuses propriétés et méthodes.

Quelques exemples d'instructions sur les feuilles :

```
'Selectionner une feuille  
Worksheets("Feuil1").Select
```

```
'Récupérer le nom de la feuille active dans une variable.  
MaFeuille = ActiveSheet.Name
```

```
'Masquer une feuille.  
Worksheets("Feuil1").Visible = False
```

```
'Supprimer une Feuille.  
Worksheets("Feuil1").Delete
```

Les exemples précédents font référence aux feuilles du classeur actif. Vous pouvez également faire référence aux feuilles des autres classeurs ouverts :

```
'Copier la Feuil2 de Classeur.xls dans un nouveau classeur  
Workbooks("Classeur.xls").Worksheets("Feuil2").Copy
```

## Les plages de cellules.

Une plage de cellules est désignée par l'objet "Range". Pour faire référence à la plage de cellule "A1:B10", on utilisera Range("A1:B10").

```
'Effacer les données et la mise en forme de la plage de cellule "A1:B10"  
Range("A1:B10").Clear
```

L'objet Range permet également de faire référence à plusieurs plages de cellules non contiguës.

```
'Sélectionner les plages de cellule "A1:B5" et "D2:F10"  
Range("A1:B5,D2:F10").Select
```

Pour faire référence à une seule cellule, on utilisera l'objet Range("Référence de la cellule) ou Cells(Numéro de ligne, Numéro de colonne).

```
'Ecrire 5 dans la cellule "A3"  
Range("A3").Value = 5  
'ou  
Cells(3, 1).Value = 5
```

Dans l'exemple suivant, nous allons recopier la plage de cellules "A1:B10" de la "Feuil1" du classeur actif dans la cellule "D5" de la "Feuil2" du classeur "Classeur2". Voici à ce que l'enregistreur de macro produirait comme code :

```
Range("A1:B10").Select  
Selection.Copy  
Windows("Classeur2").Activate  
Sheets("Feuil2").Select  
Range("D5").Select  
ActiveSheet.Paste  
Sheets("Feuil1").Select  
Application.CutCopyMode = False  
Windows("Classeur1").Activate
```

Voici maintenant le code tel qu'il pourrait être écrit sur une seule ligne de code:

```
Range("A1:B10").Copy Worksheets("Classeur2"). _  
Worksheets("Feuil2").Range("D5")
```

On peut utiliser une autre syntaxe pour faire référence à une cellule :

```
'la ligne  
Worksheets("Classeur2").Worksheets("Feuil2").Range("D5")  
'peut être remplacée par:  
Range("[Classeur2]Feuil2!D5")
```

En utilisant des variables objets (très utiles lorsque votre programme fait souvent référence aux mêmes plages de cellules), le code pourrait devenir :

```
Dim Cell As Range, Cel2 As Range  
Set Cell = Range("A1:B1")  
Set Cel2 = Worksheets("Classeur2"). _ Worksheets("Feuil3").Range("D5")  
Cell.Copy Cel2
```

VB vous permet également de changer le format des cellules (polices, couleur, encadrement ...). L'exemple suivant applique la police "courrier" en taille 10, en gras, en italique et de couleur rouge. Notez l'utilisation du bloc d'instruction **With - End With** faisant référence à l'objet Font(police) de l'objet Cell1.

```
Dim Cell As Range
Set Cell = Range("A1")
With Cell.Font
    .Bold = True
    .Italic = True
    .Name = "Courier"
    .Size = 10
    .Color = RGB(255, 0, 0)
End With
```

A partir d'une cellule de référence, vous pouvez faire appel aux autres cellules par l'instruction "Offset". La syntaxe est Range(Cellule de référence).Offset(Nombre de lignes, Nombre de colonne).

```
'Pour écrire 5 dans la cellule "B2", on pourrait utiliser :
Range("A1").Offset(1, 1) = 5
'Ecrire une valeur à la suite d'une liste de valeur dans la colonne A:
Dim NbEnreg As Integer
'NbEnreg correspond au nombre d'enregistrement de la colonne A:
NbEnreg = Range("A1").End(xlDown).Row
Range("A1").Offset(NbEnreg, 0) = 10
```

Les arguments (Nombre de lignes, Nombre de colonnes) de l'instruction Offset sont facultatifs et leur valeur par défaut est 0. La dernière ligne de code de l'exemple précédent aurait pu s'écrire :

```
Range("A1").Offset(NbEnreg) = 10
```

Nous verrons l'intérêt de cette instruction dans le cours sur les boucles.

## Les Structures de contrôle

Toutes les instructions d'un programme VBA s'articulent autour de trois structures de base :

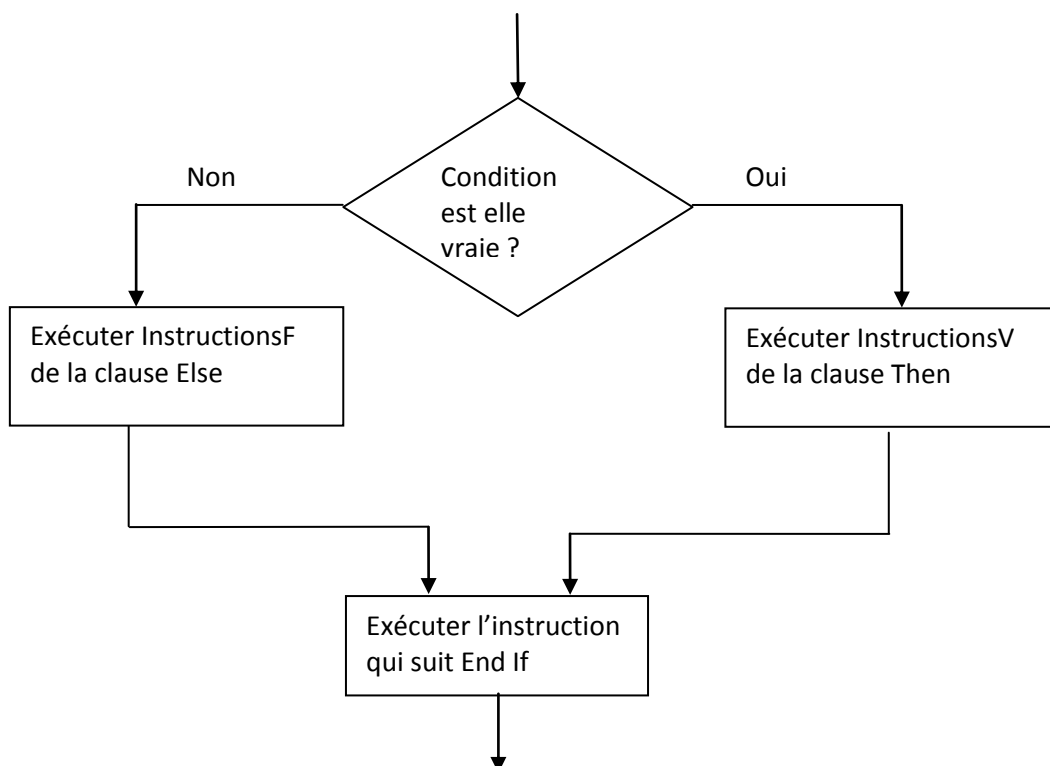
- La structure séquentielle (succession d'instructions dans un ordre chronologique bien précis).
- La structure conditionnelle (certaines instructions ne s'exécutent que sous réserve de conditions bien précises).
- La structure répétitive, ou boucle (certaines instructions peuvent être répétées un nombre fini de fois).

### La structure Conditionnelle If ... Then ... Else

L'instruction If ... Then permet d'exécuter une ou plusieurs instructions en fonction d'une ou plusieurs conditions.

La syntaxe la plus complète est :

**If Condition Then**  
**InstructionsV**  
**Else**  
**InstructionsF**  
**End If**



La structure If admet trois formats différents résumés dans le tableau ci-dessous :

Syntaxe	Commentaire
<b>If Condition Then Instruction</b>	Si Condition est Vrai Exécuter Instruction. On peut mettre plusieurs instructions après Then sur la même ligne en les séparant par deux points( :)
<b>If Condition Then InstructionsV End If</b>	Si Condition est vrai, exécuter les instructions qui se trouvent après Then, et puis passer à l'instruction qui suit End If
<b>If Condition Then InstructionsV Else InstructionsF End If</b>	Si Condition est vrai exécuter InstructionsV et passer à l'instruction qui suit End If, dans le cas contraire, exécuter InstructionsF, et passer à l'instruction qui suit End If
<b>If Condition1 Then InstructionsV1 ElseIf Condition2 Then InstructionV2 ..... ElseIf ConditionN Then InstructionsVN End If</b>	Si Condition1 est vrai exécuter InstructionsV1, et passer à l'instruction qui suit End If ; Sinon Si Condition2 est Vrai, exécuter InstructionsV2 et passer à l'instruction qui suit End If ; ..... Sinon Si ConditionN est vrai exécuter InstructionsVN, et passer à l'instruction qui suit End If ;
<b>If Condition1 Then InstructionsV1 ElseIf Condition2 Then InstructionV2 ..... ElseIf Condition n Then InstructionsVN Else InstructionsF End If</b>	Si Condition1 est vrai exécuter InstructionsV1, et passer à l'instruction qui suit End If ; Sinon Si Condition2 est Vrai, exécuter InstructionsV2 et passer à l'instruction qui suit End If ; ..... Sinon Si ConditionN est vrai exécuter InstructionsVN, et passer à l'instruction qui suit End If ; Sinon exécuter InstructionsF, et passer à l'instruction qui suit End If ;

Exemple de problèmes :

Supposons que nous disposons de la note d'un étudiant dans la cellule A1 de la feuille active, et nous voulons déterminer la décision (Echoué ou Réussi) et la mettre dans la cellule B1.

Dans le cas où la valeur de A1 est <10, la valeur de B1 est « Echoué ». Dans le cas contraire, la valeur de B1 sera « Réussi ».

Nous voulons en plus écrire « Echoué » en rouge et « Reussi » en vert.

```
Sub Decision()
    If Range("A1").Value < 10 Then
        Range("B1").Value = "Echoue"
        Range("B1").Font.Color = RGB(255, 0, 0)
    Else
        Range("B1").Value = "Reussi"
        Range("B1").Font.Color = RGB(0, 255, 0)
    End If
End Sub
```

## Structure Décisionnelle Select Case ... End select

Cette structure permet d'exécuter un des blocs d'instructions indiqués, selon la valeur d'une expression de test.

### La Syntaxe est

```
Select Case ExpressionTest
Case ListeExpression-1
    [Instructions-1]
Case ListeExpression-2
    [Instructions-2]
...
Case ListeExpression-n
    [Instructions-n]

[Case Else
    [InstructionsElse]]
End Select
```

La syntaxe de l'instruction Select Case comprend les éléments suivants :

Élément	Description
ExpressionTest	Toute expression numérique ou expression de chaîne.
ListeExpression	<p>Liste d'expressions qui peut avoir l'un des formats suivants:</p> <ul style="list-style-type: none"><li>▪ expression,</li><li>▪ expression To expression,</li><li>▪ Is opérateurcomparison expression.</li></ul> <p>Le mot clé To indique une plage de valeurs. Si vous utilisez To, la valeur la plus petite doit figurer avant To.</p> <p>Utilisez le mot clé Is avec les opérateurs de comparaison.</p>
Instruction-n	Facultatif. Une ou plusieurs instructions exécutées si ExpressionTest correspond à l'un des éléments de ListeExpression-n.
InstructionsElse	Facultatif. Une ou plusieurs instructions exécutées si ExpressionTest ne correspond à aucun élément de la clause Case.

### Remarques

Si ExpressionTest correspond à un élément de la liste ListeExpression associé à une clause Case, le bloc d'instructions qui suit cette clause est exécuté jusqu'à la clause Case suivante ou jusqu'à End Select, dans le cas de la dernière clause. Le contrôle passe ensuite à l'instruction qui suit End Select. Si ExpressionTest correspond à une expression de la liste ListeExpression

dans plusieurs clauses Case, seules les instructions qui suivent la première correspondance sont exécutées.

La clause Case Else permet d'indiquer que InstructionsElse doivent être exécutées si ExpressionTest ne correspond à aucune autre clause Case. Bien que cela ne soit pas indispensable, la présence d'une instruction Case Else dans votre bloc Select Case peut être utile lorsque ExpressionTest prend des valeurs inattendues. S'il n'y a pas d'instruction Case Else et si aucune des expressions des clauses Case ne correspond à ExpressionTest, l'exécution du programme se poursuit à partir de l'instruction qui suit End Select. Vous pouvez utiliser plusieurs expressions ou plages dans chaque clause Case. En voici un exemple :

Case 1 To 4, 7 To 9, 11, 13, Is > MaxNumber

Vous pouvez aussi indiquer des plages et des expressions multiples pour des chaînes de caractères. Dans l'exemple suivant:

Case "tout", "noix" To "soupe", TestItem

Case correspond aux chaînes de caractères qui sont absolument identiques à tout, aux chaînes comprises entre noix et soupe dans l'ordre alphabétique, ainsi qu'à la valeur en cours de TestItem.

Les instructions Select Case peuvent être imbriquées. À chaque instruction Select Case doit correspondre une instructions End Select.

## Structures d'itération ou boucles

Les structures d'itération ou boucles permettent de répéter l'exécution d'un même groupe d'instructions (appelé généralement corps de la boucle) un certain nombre de fois.

VBA fournit deux catégories de structures d'itération :

Les instructions Do...Loop qui permettent d'exécuter le corps de la boucle un nombre de fois indéfini, tant qu'une condition a la valeur True ou jusqu'à ce qu'elle prenne la valeur True.

L'instruction For ... Next qui permet de répéter l'exécution du corps de la boucle un nombre de fois connu à l'avance.

### Instructions Do...Loop

Ces instructions répètent l'exécution d'un bloc d'instructions aussi longtemps qu'une condition est vraie (True) ou jusqu'à ce qu'une condition devienne vraie (True).

Ces instructions ont deux formats différents:

Format 1	Format 2
Do [{While   Until} <i>condition</i> ]  [ <i>Instructions</i> ] Loop	Do  [ <i>instructions</i> ] Loop [{While   Until} <i>condition</i> ]

Le format 1 englobe deux syntaxes différentes :

Syntaxe 1	Syntaxe 2
Do [ While <i>condition</i> ]  [ <i>Instructions</i> ] Loop	Do [Until <i>condition</i> ]  [ <i>Instructions</i> ] Loop

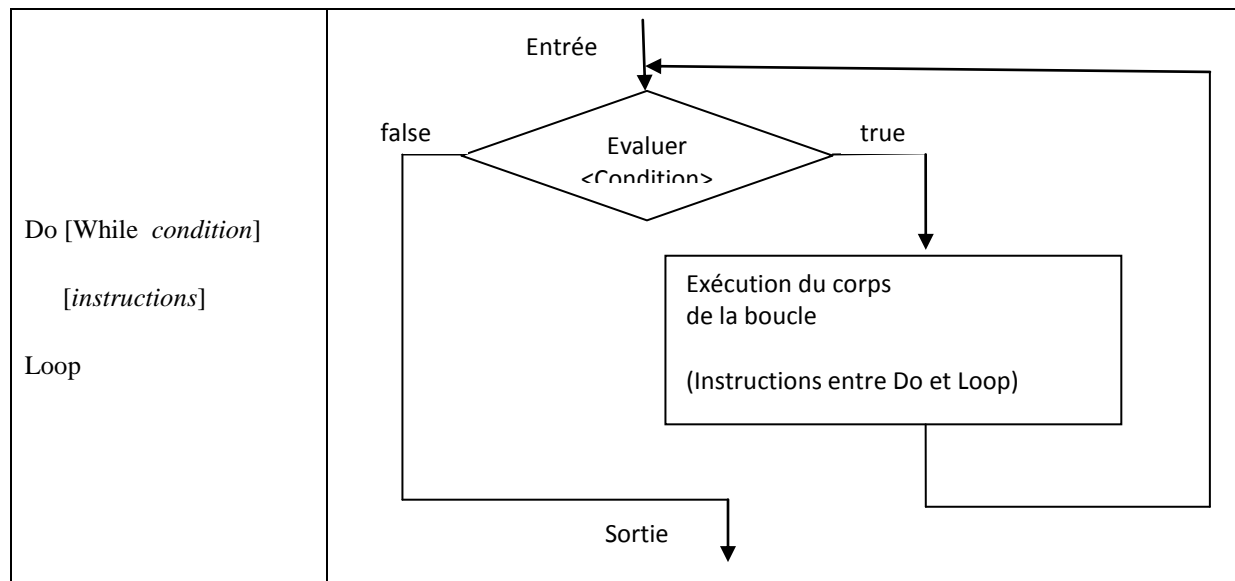
Le format 2 englobe également deux syntaxes différentes :

Syntaxe 3	Syntaxe 4
Do  [ <i>instructions</i> ] Loop [While <i>condition</i> ]	Do  [ <i>instructions</i> ] Loop [Until <i>condition</i> ]





## Syntaxe 1 de l'instruction Do...Loop



La syntaxe de l'instruction Do Loop comprend les éléments suivants :

Élément	Description
<i>condition</i>	C'est une expression numérique ou expression chaîne dont l'évaluation produit la valeur Vraie (True) ou fausse (False). Si la valeur de <i>condition</i> est Null, elle est considérée comme fausse (False). Condition est facultative, il est possible d'avoir une boucle Loop sans condition
<i>Instructions</i>	Une ou plusieurs instructions qui constituent le corps de la boucle.

### Exécution de l'instruction Do While ...Loop

Lorsque le système rencontre cette instruction, il commence par évaluer la condition. Si le résultat d'évaluation est true, le système exécute le corps de la boucle, puis réévalue de nouveau la condition, si le résultat est encore true il réexécute de nouveau le corps de la boucle et évalue encore une autre fois la condition, et ainsi de suite jusqu'à ce que le résultat d'évaluation de la condition produise une valeur false.

Si le résultat d'évaluation de la condition est dès le départ false, le système n'exécute aucune fois le corps de la boucle, et passe immédiatement à l'exécution de l'instruction qui suit Loop.

Si l'évaluation de la condition produit toujours un résultat true, le système exécutera indéfiniment le corps de la boucle, à moins qu'il y ait un dépassement de capacité. On dit que nous avons affaire à une boucle infinie. Pour arrêter une telle boucle, il faut appuyer sur ESC ou sur CTRL+ATTN.

Exemple de programme utilisant la structure Do While ... Loop

```

Sub BoucleWhile1()
    Dim Nombre AS Integer, Compteur AS Integer
    Compteur = 0
    Nombre = 20
    Do While Nombre > 10
        Nombre = Nombre - 1
        compteur = counter + 1
    Loop
    MsgBox "La boucle a effectué " & Compteur & " itérations."
End Sub

```

### Sortie d'une instruction Do...Loop à l'intérieur de la boucle : Instruction Exit Do

Il est possible de quitter une instruction Do...Loop avec une instruction Exit Do. Par exemple, pour sortir d'une boucle sans fin, utilisez l'instruction Exit Do dans le bloc d'instructions True d'une instruction If...Then...Else ou d'une instruction Select Case.

Dans l'exemple suivant, Nombre prend une valeur créant une boucle sans fin. L'instruction If Nombre < 10 Then Exit Do teste la valeur de Nombre , lorsqu'il devient inférieur à 10, Exit Do est exécutée, et le contrôle est passée à l'instruction qui suit Loop. Cela permet d'éviter une boucle sans fin.

```

Sub BoucleWhileAvecExit()
    Dim Nombre AS Integer, Compteur AS Integer
    Compteur = 0
    Nombre = 25
    Do While Nombre <> 10
        Nombre = Nombre - 2
        Compteur = Compteur + 1
        If Nombre < 10 Then Exit Do
    Loop
    MsgBox "La boucle a effectué " & Compteur & " itérations."
End Sub

```

### Instruction For...Next

Cette instruction permet de répéter l'exécution d'un même groupe d'instructions un nombre de fois connu à l'avance.

Sa Syntaxe est

```

For Compteur = Debut To Fin [Step Pas]
    [Instructions]
Next [Compteur]

```

La syntaxe de l'instruction For...Next comprend les éléments suivants :

Élément	Description
<i>Compteur</i>	Variable numérique qui joue le rôle de compteur de la boucle.
<i>Debut</i>	Valeur initiale de <i>Compteur</i> .

<i>Fin</i>	Valeur finale de <i>Compteur</i> .
<i>Pas</i>	Facultatif. Valeur d'incrémentation de <i>Compteur</i> après chaque exécution de la boucle. Si aucune valeur n'est indiquée, l'argument <i>Pas</i> prend par défaut la valeur 1.
<i>Instructions</i>	Une ou plusieurs instructions entre For et Next à exécuter le nombre de fois indiqué.

### Remarques

L'argument « *Pas* » peut être positif ou négatif. Sa valeur contrôle l'exécution de la boucle de la façon suivante :

Valeur de « <i>Pas</i> »	Condition d'exécution de la boucle
$\geq 0$	La boucle s'exécute Tant Que $\text{Compteur} \leq \text{Fin}$
$> 0$	La boucle s'exécute Tant Que $\text{Compteur} \geq \text{Fin}$

L'instruction For...Next est équivalente à l'instruction Do While ... Loop ci-dessous :

Cas où $\text{Pas} \geq 0$	Cas où $\text{Pas} < 0$
Compteur = Debut	Compteur = Debut
Do While Compteur $\leq$ Fin	Do While Compteur $\geq$ Fin
Instructions	Instructions
Compteur = Compteur + Pas	Compteur = Compteur + Pas
Loop	Loop

**Conseil** Toute modification de la valeur de *Compteur* à l'intérieur de la boucle risque de rendre la lecture et la correction des erreurs du programme plus difficiles.

Vous pouvez également placer des instructions Exit For pour quitter la boucle à tout moment. L'instruction Exit For est souvent placée après l'évaluation d'une condition (If...Then, par exemple) ; elle passe la main à l'instruction située immédiatement après l'instruction Next.

Vous pouvez imbriquer des boucles For...Next en plaçant une boucle à l'intérieur d'une autre. Vous devez utiliser des noms de variable (*Compteur*) distincts pour chaque boucle, sur le modèle de la construction suivante :

```
For I = 1 To 10
    For J = 1 To 10
        For K = 1 To 10
            ...
        Next K
    Next J
Next I
```

