

Introduction à Visual Basic pour Microsoft Excel



Mathieu Boudreault

©2002 Mathieu Boudreault
20 novembre 2002

Excel, Visual Basic et le logo de Office sont des marques déposées de Microsoft Corporation

Table des matières

Introduction	5
Introduction à Microsoft Excel	6
Qu'est-ce que Microsoft Excel ?	6
Définitions	6
Interface	6
Barres d'outils et de menus	7
Fenêtre de saisie	7
Feuille de calcul	8
Barre d'état	8
Construction d'un tableau simple	9
Formatage de cellules et de plages de cellules	9
Trucs pour l'entrée de données	10
Effectuer des calculs avec Excel	11
Saisie d'équations ou de formules	11
Truc pour la saisie de formules répétitives	12
Références relatives, absolues et mixtes	13
Un autre truc pour l'entrée d'équations : Nommer des plages	14
Insertion de fonctions	15
Outils complémentaires d'Excel : le solveur	16
Installation du solveur	16
Utiliser le solveur	16
Exemple d'application	18
Concepts fondamentaux de la programmation orientée objet	20
Définitions	20
Étapes de base dans la conception d'un programme	20
Programmer en Visual Basic pour Excel	22
Objets propres à Microsoft Excel	22
Bâtir une macro simple	23
Gestion des cellules et des plages de cellules : objets et méthodes de base	25
<i>Range</i>	25
<i>Columns</i> ou <i>Rows</i>	25
<i>Selection</i>	26
<i>ActiveCell</i>	26
<i>Offset</i>	26
<i>Cells</i>	27
<i>Activate</i>	27
<i>Select</i>	27
Remarque	27
Truc pour la manipulation répétitive du même objet	28
Gestion des feuilles de calculs et des classeurs : objets et méthodes de base	29
<i>Workbooks</i>	29
<i>ActiveWorkbook</i>	30
<i>Worksheets</i> ou <i>Sheets</i>	30
<i>ActiveSheet</i>	31
Ouverture d'un classeur <i>Excel</i> – Méthode <i>Open</i>	31
Création d'un nouveau classeur <i>Excel</i> – Méthode <i>Add</i>	31

Sélection (ou activation) d'un classeur ou d'une feuille de calcul – Méthodes <i>Activate</i> et <i>Select</i>	31
Sauvegarde d'un classeur <i>Excel</i> – Méthodes <i>SaveAs</i> et <i>Save</i>	32
Fermeture d'un classeur <i>Excel</i> – Méthode <i>Close</i>	32
Déclaration de variables.....	33
Rappel provenant des sections précédentes.....	33
Déclaration de variables scalaires	33
Déclaration de variables matrices.....	34
Assigner et obtenir des valeurs d'une variable matrice.....	35
<i>Redim</i>	36
Création de routines et de fonctions personnelles	36
Enregistrer une macro	36
Création de routines personnelles	38
Appel de routines personnelles	39
Création de fonctions personnelles	39
Appel de fonctions personnelles.....	40
Programmation conditionnelle.....	41
Créer une condition	42
Programmation conditionnelle simple.....	43
Programmation conditionnelle par cas.....	44
Programmation itérative	44
Itération simple (<i>For... Next</i>).....	45
Itération conditionnelle (<i>While... Wend</i>).....	45
Gestion des boîtes de dialogue.....	46
Boîte de dialogue <i>InputBox</i>	46
Boîte de dialogue <i>MsgBox</i>	47
Une vue d'ensemble de l'éditeur Visual Basic	50
Trucs de débogage	50
<i>Commande Exécuter</i>	50
<i>Exécuter Pas à pas</i>	51
Points d'arrêt	52
Fenêtre espions.....	53
Messages d'erreurs.....	54
Erreurs de compilation.....	54
Erreurs d'exécution.....	54
Erreurs logiques	55
Suite de l'exemple – Construction du tableau de primes	56
Éléments de programmation avancée en Visual Basic pour Excel	59
Objet <i>WorksheetFunction</i>	59
Utiliser une fonction publique Visual Basic créée par le programmeur à l'intérieur d'un classeur.....	60
Créer une fonction publique en 3 étapes simples.....	60
Exemple complet.....	61
Déclaration de variables de type <i>Object</i>	61
Déclaration de variables de type <i>Object</i>	62
Assignation dans une variable de type <i>Object</i>	62
Méthode <i>InputBox</i>	62
Anciens travaux pratiques de l'année 2000	64

Travail pratique # 1	64
Contexte	64
Solution.....	64
Travail pratique # 2	66
Contexte	66
Solution.....	67
Travail pratique # 3	68
Contexte	68
Solution.....	69
Travail pratique # 4	72
Contexte	72
Solution.....	72
Travail pratique # 5	75
Contexte	75
Solution.....	75
Références	78

Introduction

Bien que le logiciel Microsoft Excel soit un outil très puissant aux capacités touchant plusieurs domaines tels que les mathématiques, la finance, l'ingénierie, etc., il ne peut malheureusement tout faire. C'est grâce à la programmation en Visual Basic pour Excel que les utilisateurs les plus avancés pourront adapter Microsoft Excel à leur besoin. Il est alors possible de combiner plusieurs des outils d'Excel et de les améliorer, de les compléter, créer plusieurs commandes et outils supplémentaires, etc. En actuariat, Visual Basic pour Excel intervient dans tous les domaines connexes : régime de retraite, assurance générale, assurance vie, et même en recherche.

Ce document se veut une présentation des fonctions de base d'Excel et d'une introduction à la programmation en Visual Basic pour Excel. Vous verrez qu'une fois les connaissances de base acquises en programmation, la marche n'est pas si haute pour passer de Visual Basic à Visual Basic pour Excel. Il a été bâti de telle sorte qu'il peut agir comme document de référence dans le futur.

Dans ce document, vous verrez le fonctionnement de base de Microsoft Excel, la construction de tableaux, l'utilisation de formules et de fonctions, etc. Ensuite, une fois ces notions bien maîtrisées, il est ainsi plus facile de manipuler les objets Excel de la même façon que les objets Visual Basic. Vous découvrirez ensuite la programmation en Visual Basic, l'utilisation des objets Excel, etc. Toutes les notions amenées sont accompagnées d'exemples. Vous retrouverez aussi à la fin, quelques exemples complets et les travaux pratiques de l'année 2000 (avec solutions).

Tout au long du document, si vous voulez en savoir plus, je vous invite à consulter l'aide de Microsoft Excel et Visual Basic pour Excel. Afin de mieux avancer de façon autonome en Visual Basic pour Excel, il est fortement recommandé de bien comprendre les notions d'objets, méthodes et propriétés. De cette façon, il est ainsi beaucoup plus facile d'utiliser de nouveaux objets autrefois inconnus pour soi.

Je tiens aussi à remercier MM. Yan Bergeron, Simon Gamache et Frédérick Guillot pour l'aide qu'ils m'ont apportée dans la complétion du document. Leur contribution a été grandement appréciée.

Introduction à Microsoft Excel

Qu'est-ce que Microsoft Excel ?

Un tableur ou chiffrier électronique est une table géante qui permet de calculer des données, les analyser et les organiser dans des tableaux. En termes informatiques, Microsoft Excel est donc un logiciel qui permet de gérer une multitude de feuilles de calculs. Une feuille de calcul est un ensemble de données et d'équations. Par exemple, la construction d'une facture dans Excel pourrait constituer une feuille de calcul. En actuariat, un logiciel comme Excel peut servir l'actuaire dans plusieurs domaines et applications possibles tels que la gestion des tables de mortalité et des probabilités de décès, détermination des réserves, simulation, optimisation, etc.

Définitions

Cellule (Cell) : Élément d'un tableau.

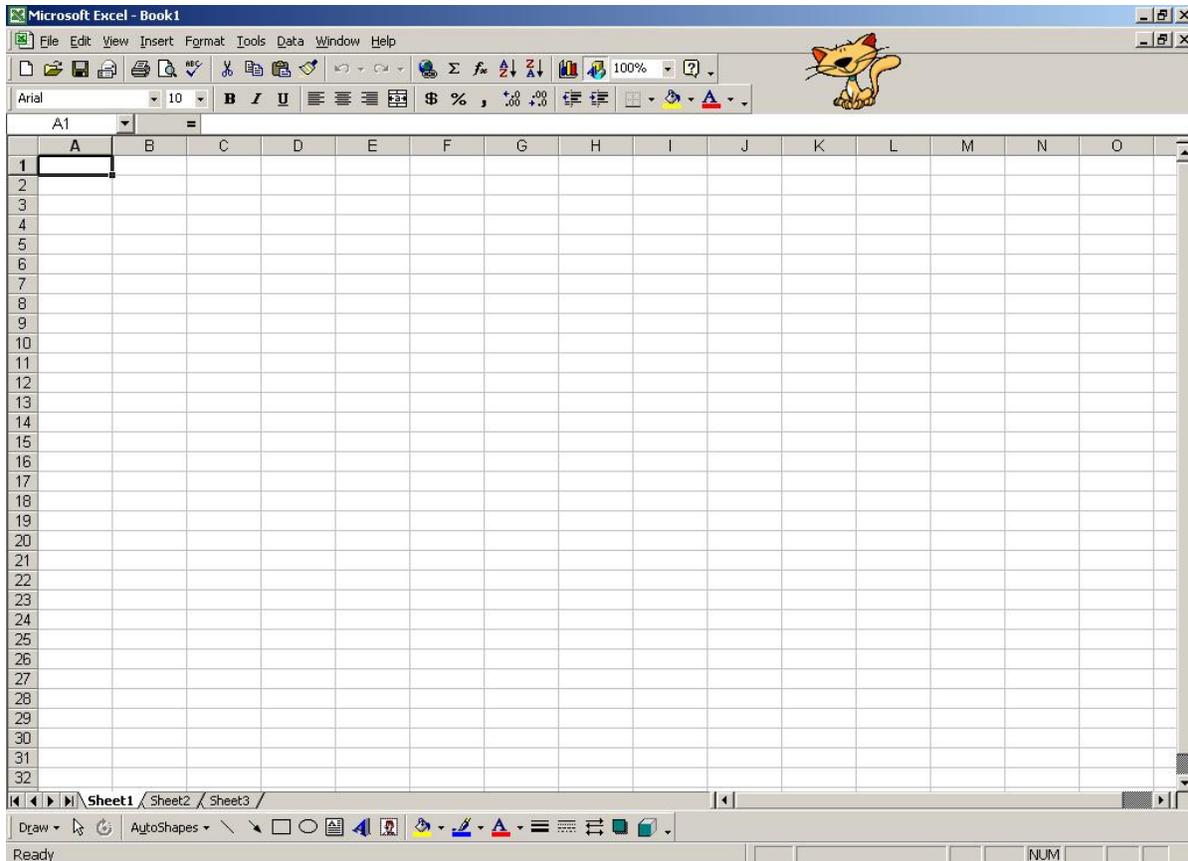
Classeur (Workbook) : Ensemble de feuilles de calculs. Un fichier Excel (.xls) est un classeur.

Feuille de calcul (Worksheet) : Ensemble de données et d'équations organisées en tableaux.

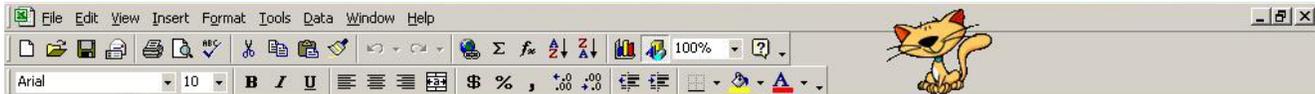
Plage (Range) : Ensemble de cellules.

Interface

Regardons ensemble la construction de l'interface principale d'Excel.



Barres d'outils et de menus



Les barres d'outils et de menus permettent d'accéder aux principales fonctionnalités d'Excel.

En haut, on retrouve les différents menus d'Excel.

- **Fichier (*File*)** : Fonctionnalités de gestion de fichier. Par exemple, ouverture, fermeture, sauvegarde, mise en page, etc.
- **Édition (*Edit*)** : Fonctionnalités d'édition (modification) des données. On y retrouve par exemple les fonctions très utiles Copier, Couper, Coller et Effacer.
- **Affichage (*View*)** : Fonctionnalités permettant de modifier l'affichage du fichier en cours.
- **Insertion (*Insert*)** : Fonctionnalités d'ajout d'objets à l'intérieur du classeur. On peut ajouter une image, un graphique, une fonction Excel, etc.
- **Format (*Format*)** : Fonctionnalités de formatage d'un élément d'Excel. Le formatage peut être de modifier la police et la taille du texte, la couleur de fond de la cellule, etc. En bref, il s'agit des outils qui permettent de modifier l'apparence d'un élément d'Excel.
- **Outils (*Tools*)** : Fonctionnalités permettant d'accéder aux différents outils de Microsoft Excel tels que le correcteur d'orthographe, création de macros, solveur (outil d'optimisation), etc.
- **Données (*Data*)** : Fonctionnalités de gestion des données, comme le tri des données, etc.

Ensuite, on retrouve la barre d'outil principale et de formatage. Avec ces boutons, il est possible d'accéder aux fonctionnalités les plus communes d'Excel.

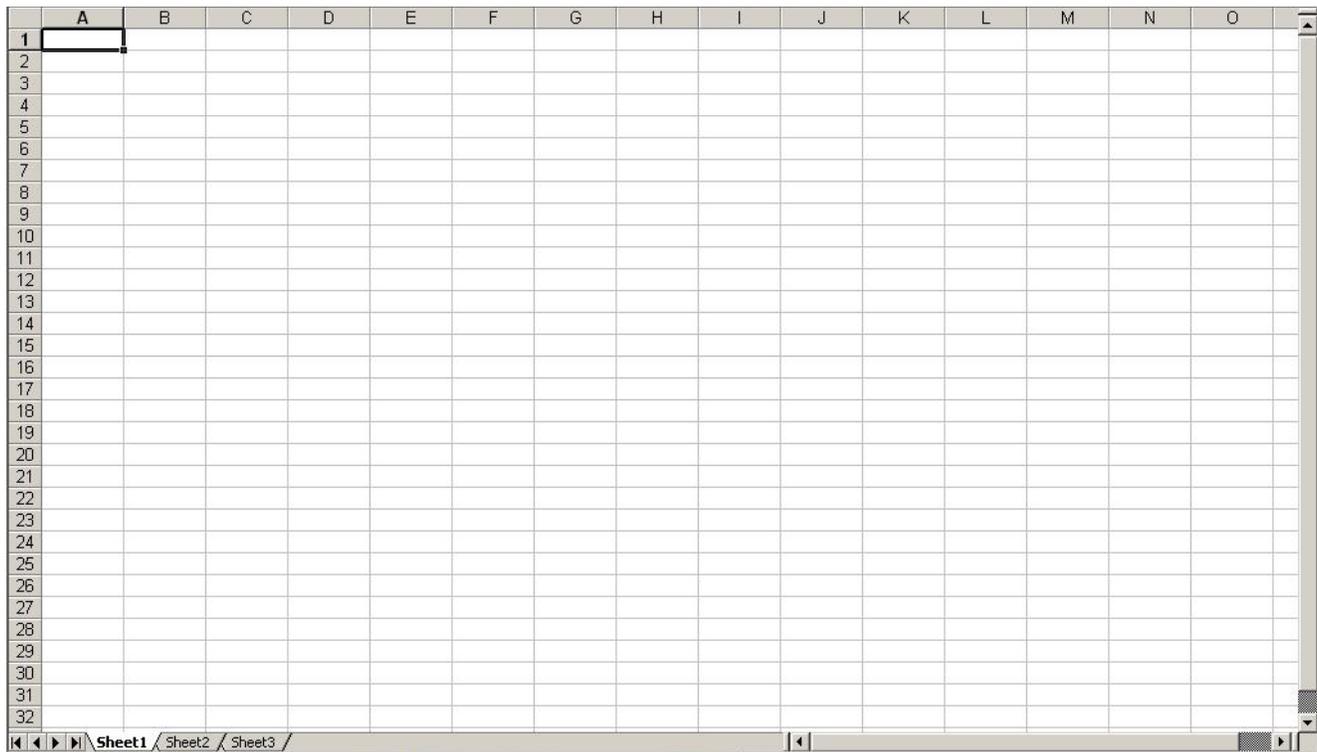
D'autres barres d'outils sont aussi disponibles. Il suffit d'aller dans le menu Affichage et ensuite pointer vers Barre d'outils.

Fenêtre de saisie



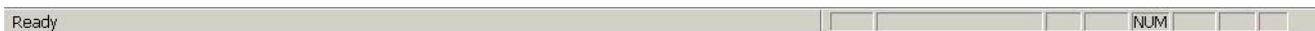
C'est à cet endroit que les données et les formules peuvent être saisies. On remarque à gauche l'adresse de la cellule. L'adresse d'une cellule est toujours composée d'une lettre suivie d'un chiffre. La lettre représente la colonne et le chiffre représente la ligne. Par exemple, G10 correspond à la cellule située à la 7^e colonne et 10^e ligne.

Feuille de calcul



Voici l'interface principale d'une feuille de calcul. C'est à l'intérieur de cette zone que sont situées les données et les équations. Dans le bas à gauche, les onglets permettent de basculer entre les différentes feuilles de calcul. Il est possible d'ajouter des feuilles en cliquant avec le bouton droit de la souris et en choisissant Insertion. Il suffit ensuite de sélectionner Feuille. Vous pouvez aussi renommer vos feuilles de calcul en double-cliquant sur le nom actuel de celle-ci.

Barre d'état



Différents messages sont affichés dans la barre d'état selon le contexte.

Construction d'un tableau simple

Nous allons construire ensemble le tableau suivant :

x	Probabilité de décès dans la prochaine année d'un individu d'âge x
35	0.015
36	0.020
37	0.025
38	0.030
39	0.035
40	0.040

Il s'agit en fait d'entrée de données. La cellule supérieure gauche de ce tableau doit être en A1. À l'état brut, votre tableau devrait ressembler à celui-ci.

	A	B	C	D	E	F	G	H
1	x	Probabilité de décès dans la prochaine année d'un individu d'âge x						
2		35	0,015					
3		36	0,02					
4		37	0,025					
5		38	0,03					
6		39	0,035					
7		40	0,04					
8								
9								

Formatage de cellules et de plages de cellules

Il est possible de modifier l'apparence du tableau et de chacun des éléments de celui-ci. Il y a trois différentes façons pour y parvenir : (1) les menus, (2) les barres d'outils ou (3) le bouton droit ou menu contextuel. Tout d'abord, il faut sélectionner la cellule ou l'ensemble de cellules dont on veut modifier l'apparence. Ensuite, vous n'avez qu'à choisir une des trois façons suivantes pour modifier l'allure des cellules.

1. Menu Format;
2. Deuxième barre d'outil;
3. En cliquant avec le bouton droit de la souris dans la zone sélectionnée.

Il est possible de modifier le format du nombre, l'alignement du texte à l'intérieur de la cellule, la police et la taille du texte, la bordure autour de la plage de cellule ainsi que la couleur de fond. On peut appliquer ces modifications à une cellule ou à un ensemble de cellules.

Exercice : Faites en sorte que le tableau moche initial ressemble à celui-ci.

	A	B	C	D	E	F
1	x	Probabilité de décès dans la prochaine année d'un individu d'âge x				
2	35	0,015				
3	36	0,020				
4	37	0,025				
5	38	0,030				
6	39	0,035				
7	40	0,040				
8						
9						
10						
11						

Solution (en utilisant une des 3 façons)

Cellules A1 jusqu'à B1

Alignement : Horizontal centré, Vertical centré, Retour à la ligne automatique

Police : Arial, 12 pts, Gras

Bordure : Contour 3 pts

Arrière-plan : Couleur cyan

Cellules A2 jusqu'à B7

Alignement : Horizontal centré

Police : Arial, 12 pts

Bordure : Contour 3 pts, Ligne au centre : 1 pt

Arrière-plan : Couleur jaune

Cellules B2 jusqu'à B7

Nombre : Nombre de décimales : 3

Modifiez la largeur des colonnes (ou la hauteur des lignes) en amenant la souris sur le côté gauche (ou haut) de la feuille de calcul. Le curseur devrait changer de forme, vous indiquant que vous pourrez ajuster la largeur ou hauteur d'une colonne ou ligne.

Trucs pour l'entrée de données

Lorsqu'une séquence se présente à l'intérieur des données, Excel est capable d'extrapoler et de compléter la séquence. Par exemple, ici, au lieu d'entrer 35, 36, 37,..., 40, il suffisait d'entrer 35 et 36 à l'intérieur des cellules A2 et A3 et de compléter la séquence à l'aide de la poignée. Voici comment :

1. Entrez 35 et 36 à l'intérieur des cellules A2 et A3 et sélectionnez la plage A2:A3. Note : pour Excel, le deux-points (:) signifie «jusqu'à». Exemple : A1:G8 correspond au tableau formé des cellules allant de A1 (coin supérieur gauche de la plage) jusqu'à G8 (coin supérieur droit de la plage).

2. Cliquez sur la poignée de la sélection et maintenez le bouton gauche de la souris enfoncé. Voici la fameuse poignée de sélection :



3. Glissez la souris jusqu'à ce que le nombre 40 apparaisse.

Pour entrer les probabilités de décès, puisque le pas de la séquence est de 0.005, il est aussi possible d'utiliser ce truc. Il y a aussi une autre façon de procéder pour entrer les probabilités de décès. Répétez les étapes 1 et 2 précédentes pour la colonne B (probabilités de décès). Au lieu de glisser la poignée jusqu'en bas, double-cliquez sur celle-ci. Note importante : le tout dernier truc fonctionne **seulement si** des données sont présentes dans la colonne à sa gauche.

Effectuer des calculs avec Excel

Une fois que les données sont bien organisées, il est souvent utile d'effectuer des calculs sur ceux-ci. Il y a deux façons d'exécuter des calculs avec les cellules :

- Équations ou formules;
- Fonctions.

Saisie d'équations ou de formules

Observons ensemble la première façon d'exécuter des calculs.

Syntaxe

=CELLULE/NOMBRE Opérateur mathématique CELLULE/NOMBRE Opérateur mathématique CELLULE/NOMBRE...

CELLULE : Correspond à l'adresse d'UNE cellule. Important : il est impossible d'exécuter une équation avec une plage de cellules.

Opérateur mathématique : Les opérateurs sont :

- Addition : +
- Soustraction : -
- Multiplication : *
- Division : /
- Exposant : ^
- Parenthèses : ()

Note : La priorité des opérations est respectée.

Exemple # 1

Dans la cellule E2, entrez : =2+3

Le résultat, 5 s'affichera dans la cellule E2.

Exemple # 2

Dans la cellule E3, entrez 10, en F3, entrez 3.

Si dans la cellule E4 vous tapez : $=14+E3^F3$, vous aurez le résultat 1014.

Note importante

Pour saisir une équation, vous pouvez entrer directement au clavier $=14+E3^F3$ ou vous pouvez entrer $=14+$ au clavier, ensuite, à l'aide des flèches du clavier, déplacer la cellule sélectionnée vers E3, ensuite saisir $^$ au clavier et ensuite choisir la cellule F3 à l'aide des flèches.

Poursuivons maintenant l'exemple principal.

Premièrement, effacez le contenu des cellules E3:F4 et ensuite, insérez deux lignes au-dessus du tableau. Il y a différentes façons d'y parvenir. Vous pouvez tout simplement déplacer le tableau vers le bas de deux lignes (couper, coller) ou aller dans le menu Insertion et choisir Ligne (répétez 2 fois pour deux lignes).

Il faut reproduire le tableau suivant

	A	B	C	D
1		Prestation de décès	10 000,00 \$	
2				
	x	Probabilité de décès dans la prochaine année d'un individu d'âge x	Prime assurance temporaire un an émis à l'individu d'âge x	
3				
4	35	0,015	150,00 \$	
5	36	0,020	200,00 \$	
6	37	0,025	250,00 \$	
7	38	0,030	300,00 \$	
8	39	0,035	350,00 \$	
9	40	0,040	400,00 \$	
10				

Il est important de savoir que si vous modifiez le contenu de la cellule C1, les primes doivent aussi se mettre à jour. Cette exigence implique que dans les cellules C4:C9 il doit y avoir une équation. Par exemple, en C4, l'équation à saisir est : $=C1*B4$. En C5, il faut saisir, $=C1*B5$, etc. Un peu long et redondant n'est-ce pas ?

Truc pour la saisie de formules répétitives

Un peu comme nous l'avons fait dans le cas où une séquence se présentait dans les données, il est aussi possible d'appliquer la même équation à un ensemble de cellules à l'aide de la poignée de sélection. Si en C4, vous avez $=C1*B4$, cliquez deux fois sur la poignée et vous aurez :

	A	B	C	D	E
1		Prestation de décès	10 000,00 \$		
2					
3	x	Probabilité de décès dans la prochaine année d'un individu d'âge x	Prime assurance temporaire un an émis à l'individu d'âge x		
4	35	0,015	150,00 \$		
5	36	0,020	- \$		
6	37	0,025	#VALUE!		
7	38	0,030	4,50 \$		
8	39	0,035	- \$		
9	40	0,040	#VALUE!		
10					
11					

Comme vous pouvez le constater, le calcul de la prime a fonctionné que pour la cellule C4. Pourquoi ? Il est très important de comprendre la prochaine section.

Références relatives, absolues et mixtes

Lorsque vous avez entré en C4, $=C1*B4$, vous avez en quelque sorte dit à Excel : «Prends la cellule immédiatement à ta gauche (B4) et multiplie la par la cellule qui est 3 lignes plus haut (C1)». C'est effectivement une autre façon d'interpréter nos désirs. Par contre, en appliquant la même équation jusqu'à C9, Excel a appliqué aussi le même raisonnement pour chacune des cellules comprises dans la plage C4:C9. Exemple, en C9 il s'est dit : «Je dois prendre la cellule immédiatement à ma gauche (B9) et la multiplier par la cellule qui est 3 lignes plus haut (C6)». Cette façon de raisonner est ce qu'on appelle une référence relative.

Pour notre exemple, nous voulons le raisonnement suivant : «Prends la cellule immédiatement à ta gauche (B4) et multiplie la par la cellule C1». Ici, B4 est une référence relative et C1 est une référence absolue. Comment faire en sorte qu'Excel reconnaisse les références absolues ? Au lieu d'entrer dans la cellule C4 l'équation $=C1*B4$, il suffit d'entrer $=$C$1*B4$. Ensuite, répétez le truc du double-clic sur la poignée. Voilà !

Lors de la création d'équations et de l'application de celles-ci sur plusieurs cellules, il est aussi possible de définir des références mixtes, c'est-à-dire définir, par exemple, la ligne d'une cellule de façon absolue et la colonne de façon relative. Nous aurions, \$C4. Si l'équation est appliquée simultanément sur plusieurs lignes et colonnes, l'équation sera toujours une fonction du contenu de la colonne C.

Exemple : Vous désirez construire une table de multiplication des 5 premiers nombres naturels. Vous devez obtenir ce résultat :

	A	B	C	D	E	F	G	H
1								
2		Table de multiplication						
3								
4		X	1	2	3	4	5	
5		1	1	2	3	4	5	
6		2	2	4	6	8	10	
7		3	3	6	9	12	15	
8		4	4	8	12	16	20	
9		5	5	10	15	20	25	
10								
11								

Vous devriez être assez à l'aise pour formater ce tableau. Regardons ensemble comment obtenir cette table sans avoir à écrire 25 équations différentes.

Si en C5, vous entrez $=B5*C4$ et que vous appliquez cette formule à toutes les cellules du tableau, vous obtiendrez la table suivante :

	A	B	C	D	E	F	G	H	I
1									
2		Table de multiplication							
3									
4		X	1	2	3	4	5		
5		1	1	2	6	24	120		
6		2	2	4	24	576	69120		
7		3	6	24	576	331776	2,29E+10		
8		4	24	576	331776	1,1E+11	2,52E+21		
9		5	120	69120	2,29E+10	2,52E+21	6,37E+42		
10									
11									
12									
13									

Pour obtenir le résultat voulu, il faut entrer en C5, l'équation suivante : $=\$B5*C\4 . Ici, on veut maintenir fixe la référence à la colonne B et maintenir fixe la référence à la ligne 4. Le reste doit varier selon la position. Appliquez maintenant cette équation à toutes les cellules du tableau à l'aide de la poignée.

Un autre truc pour l'entrée d'équations : Nommer des plages

Au lieu d'entrer des équations à l'aide des adresses des cellules, il est aussi possible de donner des noms à des plages de cellules et appeler les équations à l'aide de ces noms. Voici comment faire.

Supposons que nous allons nommer la plage B4:B9 «prob». Sélectionnez cette plage, puis dans la zone de nom de plage (voir illustration suivante), entrez le mot «prob».



Procédez de la même façon pour la cellule C1 que vous nommerez «prestation». Ensuite, dans la cellule C4, entrez la formule $=prob*prestation$. Vous obtiendrez le même résultat que précédemment. En plus, si vous descendez la formule à l'aide de la poignée, vous obtiendrez les primes exactes pour tout âge.

Insertion de fonctions

Il y a aussi une deuxième façon d'effectuer des calculs avec Excel : l'utilisation de fonctions. Une fonction est une formule prédéfinie qui effectue un calcul à l'aide des arguments de la fonction. Pour utiliser une fonction on doit utiliser la syntaxe suivante.

Syntaxe

=NOMFONCTION(argument1 ; argument2 ; argument3 ;...)

On peut aussi accéder à la liste des fonctions ainsi qu'à leur description soit en allant dans le menu Insertion, Fonction ou en appuyant sur la touche *fx* dans la barre d'outils.

Les arguments d'une fonction représentent ce que la fonction a besoin pour retourner une valeur. Ils peuvent être de différents types : nombre, logique (booléen), etc.

Dans Excel, ces arguments peuvent aussi faire référence à une cellule ou à une plage de cellules. Il est aussi possible d'utiliser les références relatives et absolues.

Exemple

Les notes de 50 étudiants sont entrées dans les cellules A5:A54. On veut obtenir la moyenne et l'écart type.

Pour calculer la moyenne, au lieu d'entrer la formule $=(A5+A6+A7+\dots+A54)/50$, nous utiliserons la fonction MOYENNE.

=MOYENNE(A5:A54)

Même principe pour calculer l'écart-type des notes de la classe. Nous utiliserons la fonction =ECARTYPE(A5:A54)

Remarques

- Il est possible d'imbriquer des fonctions à l'intérieur d'une autre fonction.
- Les opérations mathématiques peuvent être utilisées à l'intérieur d'une fonction et avec une fonction. Exemple : Calculer la variance des notes du groupe. On entre la fonction =ECARTYPE(A5:A54)^2.

Voici un exemple combinant les deux remarques.

On désire calculer la probabilité qu'un étudiant pris au hasard ait plus que 70% à son examen. On suppose que la distribution des notes est normale. La fonction à entrer est =1-LOI.NORMALE(0.70 ; MOYENNE(A5:A54) ; ECARTYPE(A5:A54) ; VRAI)

Ici, la fonction LOI.NORMALE retourne la fonction de répartition ou de densité d'une loi normale. Dans cet exemple, il s'agit de la fonction de répartition (VRAI) d'une loi normale, évalué à 0.70, avec paramètres $\mu = \text{MOYENNE}(A5:A54)$ et $\sigma = \text{ECARTYPE}(A5:A54)$.

Outils complémentaires d'Excel : le solveur.

À l'intérieur de Microsoft Excel, il y a plusieurs outils permettant de calculer et d'analyser des données. Ces outils sont la création de graphiques, de cartes géographiques, outils d'optimisation, d'analyse des données en régression, création d'histogrammes, résolution de problèmes à l'aide de la transformée de Fourier, etc. Nous regarderons ensemble un outil très intéressant en mathématiques et en actuariat dans la résolution de problèmes : le solveur.

Le solveur du logiciel Microsoft Excel est un utilitaire permettant de résoudre efficacement et rapidement certains problèmes d'optimisation en effectuant une approximation de la solution par le biais d'algorithmes numériques. Les problèmes de type linéaires sont traités à l'aide de l'algorithme du simplexe et la méthode de *Branch-and-Boundmise*, tandis que les problèmes de type non linéaires sont résolus à l'aide de la méthode du *Generalized Reduced Gradient*.

Son fonctionnement est très simple. En termes clairs, il adapte les valeurs des cellules que vous souhaitez modifier (cellules variables), et retourne le type de résultat spécifié (maximum, minimum, etc.) à partir des formules des cellules cibles et des contraintes spécifiées. Le solveur peut notamment être utilisé dans la recherche des extrema d'une fonction complexe, pour résoudre un système d'équation linéaire, pour estimer des paramètres ou pour résoudre un problème algébrique.

Installation du solveur

Si le mot "Solveur" ne figure pas dans la liste déroulante du menu "Outils", il vous faudra installer le solveur. Voici la procédure pour installer le solveur :

- (1) Choisir "Macro complémentaire" dans le menu "Outils" ;
- (2) Cocher solveur;
- (3) Appuyer sur "OK".

Utiliser le solveur

Pour résoudre un problème avec le solveur, il faut commencer par monter un classeur Excel en reliant certaines cellules par les formules de votre problème. Par exemple, si on cherche la solution du système d'équations suivant :

$$i) 2x + y + z = 3$$

$$ii) x - y - z = 3$$

$$iii) 3x + 2y + 3z = 3$$

Il pourrait être intéressant de construire ce type de feuille Excel :

	A	B	C	D
1				
2				
3	Variables :			
4		x :	1	
5		y :	1	
6		z :	3	
7				
8	Equations (Retrouvées avec des formules)			
9				
10				
11		2x+y+z :	6	
12		x-y-z :	-3	
13		3x+2y-3z :	-4	
14				
15				
16				
17				

Remarque

Initialement, on a posé que $x = 1, y = 1$ et $z = 3$. Ce ne sont que des valeurs fictives qui seront par la suite ajustées par le solveur pour satisfaire au système d'équations linéaire.

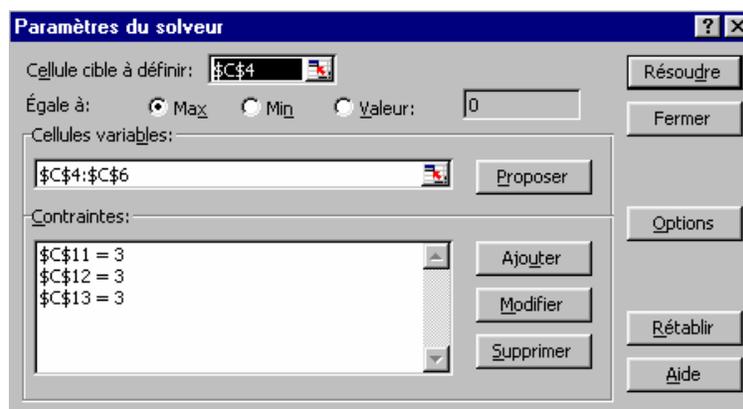
La seconde étape consiste à décortiquer le problème en déterminant les cellules qui sont :

- (1) À optimiser (minimiser, maximiser ou égaliser);
- (2) Variables;
- (3) Sujettes à une contrainte d'optimisation.

Dans notre exemple, la cellule C4 pourrait être arbitrairement choisie comme une cellule à maximiser, les cellules C4, C5 et C6 seraient les cellules variables (celles que le solveur devra réajuster pour que le système d'équations soit respecté) et les cellules C11, C12 et C13 seront sujettes à des contraintes (ces cellules doivent être égales à 3 pour que le système d'équation soit satisfait). Étant donné les contraintes mises en place, le fait de maximiser arbitrairement la cellule C4 ne modifie en rien le résultat.

Lorsque la feuille Excel est décortiquée, il ne reste plus qu'à entrer les paramètres dans le solveur et à exécuter ce dernier.

Dans notre exemple, voici comment il faudrait entrer les paramètres du solveur :



Voici maintenant l'allure de la feuille Excel après le travail du solveur :

	A	B	C	D
1				
2				
3	Variables :			
4		x :	2,00000033	
5		y :	-1,20000006	
6		z :	0,19999993	
7				
8	Equations (Retrouvées avec des formules)			
9				
10				
11		2x+y+z :	3	
12		x-y-z :	3,0000001	
13		3x+2y-3z :	3	
14				

La solution du système d'équations est sans aucun doute $x = 2$, $y = -1,2$ et $z = 2$.

Attention : Lorsque le problème est plus complexe, le solveur peut ne pas converger vers la solution à la première exécution. Dans ce cas, on peut exécuter le solveur une ou plusieurs autres fois ou modifier le nombre d'itérations maximal dans les options du solveur.

Exemple d'application

Un dentiste souhaite modéliser le nombre d'obturations dans la bouche d'un patient en fonction de l'âge de celui-ci. Lors de son avant-midi de travail, il a observé 5 bouches et inscrit ses observations dans le tableau suivant :

Patient i	Nombre d'obturations dans la bouche du patient i (Y_i)	Âge du patient i (X_i)
1	3	13
2	5	28
3	2	15
4	7	22
5	2	10

Le dentiste suppose que

$$\hat{Y}_i = \alpha X_i + \beta \ln(X_i) + \lambda \sqrt{X_i},$$

où α , β et λ sont des constantes. Le dentiste souhaite déterminer la valeur de ces constantes de sorte à ce que la somme des carrés des erreurs $(\sum_{i=1}^5 (Y_i - \hat{Y}_i)^2)$ soit minimisée. Le dentiste construit donc le fichier Excel suivant (et pose arbitrairement que $\alpha = 2$, $\beta = 5$ et $\lambda = 7$):

	A	B	C	D	E	F
1	alpha :	2				
2	beta :	5				
3	lambda :	7				
4						
5						
6	i	Yi	Xi	[^] Yi	(Yi- [^] Yi) ²	
7	1	3	13	64,0636057	3728,76394	
8	2	5	28	109,701541	10962,4127	
9	3	2	15	70,6511344	4712,97826	
10	4	7	22	92,2881226	7274,06385	
11	5	2	10	53,6488691	2667,60568	
12						
13				Total :	29345,8244	
14						
15						

Voici ce que le dentiste a entré dans les paramètres du solveur :

Et voici le fichier Excel après l'exécution du solveur :

	A	B	C	D	E	F
1	alpha :	0,18140468				
2	beta :	-1,02407027				
3	lambda :	0,84517454				
4						
5						
6	i	Yi	Xi	[^] Yi	(Yi- [^] Yi) ²	
7	1	3	13	2,77889262	0,04888847	
8	2	5	28	6,13916282	1,29769192	
9	3	2	15	3,22118345	1,49128901	
10	4	7	22	4,7896783	4,885522	
11	5	2	10	2,12871445	0,01656741	
12						
13				Total :	7,73995881	
14						

Le dentiste a un patient de 32 ans cet après midi, selon ses observations de l'avant-midi et selon son modèle mathématique, il peut s'attendre à ce que ce patient ait 7 obturations en bouche !

Concepts fondamentaux de la programmation orientée objet

Avant de débiter la programmation en Visual Basic, il faut rappeler certains concepts importants de la programmation orientée objet.

Définitions

Objet : Élément de programmation ayant des propriétés, pouvant exécuter certaines actions et pouvant interagir avec d'autres objets. Un objet peut en contenir un autre et peut aussi appartenir à d'autres objets.

Propriété : Caractéristique ou attribut propre à un certain objet.

Méthode : Action que peut exécuter un objet.

Observons ce que pourrait être un objet sans penser à la programmation. Par exemple, une voiture. Effectivement, une voiture peut avoir certaines caractéristiques (couleur, longueur, largeur, puissance, consommation, etc.) La voiture peut aussi exécuter certaines actions (avancer, reculer, etc) et peut aussi interagir avec certains autres éléments (la voiture interagit avec la route, etc.). Une radio, qui est aussi un objet en soi ayant des propriétés, peut être contenu dans une voiture et aussi être contenu dans une maison (objet).

Collection ou classe : Ensemble d'objets ou d'autres collections.

Hiérarchie : Façon d'ordonner les objets et les collections. Exemple : ClasseSuperieure.ClasseInferieure.Objet.Propriete = True. Ici, on a que l'objet appartient à la classe inférieure, qui elle appartient à la classe supérieure. Sa propriété est fixée à *True*.

Variable : Espace en mémoire réservé pour y stocker de l'information. Une variable peut être de plusieurs types : numérique (entier, réel), booléen (ou logique), texte (ou chaîne de caractères, aussi appelé *string*).

Étapes de base dans la conception d'un programme

1. Bien comprendre le problème. La première étape de la conception d'un programme ne doit jamais impliquer le contact direct avec un ordinateur. Un important processus de réflexion et de compréhension doit se faire chez le programmeur avant de se lancer dans la conception. Il peut être utile de bien schématiser les étapes.
2. Diviser pour régner
 - a. Diviser le problème principal en sous-problèmes.
 - b. À chaque sous-problème est associé une fonction ou sous-programme. De cette façon, il devient beaucoup plus facile de vérifier l'exactitude du programme en entier. Si un bogue se présente, il est ainsi plus simple de localiser la ligne de code fautive car seulement un résultat intermédiaire s'en trouve erroné.
Note : Chacune des étapes qui suivent doit être répétée pour chaque sous-programme.
 - c. Il est alors possible de débiter la programmation.
3. Déclaration des variables qui vont être utilisées dans le programme ou sous-programme. Dans l'utilisation des variables, il y a toujours deux étapes nécessaires.

Premièrement, la déclaration qui alloue ou réserve un espace en mémoire. Deuxièmement, le stockage utilise l'espace qui lui est réservé pour conserver l'information voulue. Il est aussi fortement recommandé de donner un nom intelligent à ses variables. Un principe fondamental : le nom d'une variable devrait refléter son contenu.

4. Code principal.
 - a. Obtenir les données nécessaires au calcul;
 - b. Effectuer le calcul;
 - c. Afficher le résultat.
5. Une fois le programme et les sous-programmes bien rôdés, il est fortement recommandé de bien documenter celui-ci. L'idéal est de bien décrire le rôle de chaque sous-programme ainsi que leurs arguments. Les lignes de code moins intuitives (par exemple, les éclairs de génie) devraient aussi être bien décrites. Une question à se poser lorsqu'on documente ses routines : en lisant le code ainsi que le texte, est-ce que mes collègues seraient en mesure de bien comprendre l'idée principale de ma programmation ainsi que les principales étapes de mon code ? Évidemment, répondre par la négative à cette question devrait vous inciter à continuer la documentation.

Programmer en Visual Basic pour Excel

Tout d'abord, il convient de faire la distinction entre Visual Basic et Visual Basic pour applications. Premièrement, il s'agit du même langage de programmation. Les commandes, les structures de programmes et l'interaction entre les objets est identique. Visual Basic est un logiciel complet en soi : il permet de créer des applications complètement autonomes. Visual Basic pour applications est un complément à chacun des logiciels inclus dans la suite Microsoft Office. Par exemple, Visual Basic pour Microsoft Excel permet à l'utilisateur de créer des petites applications qui interagissent avec un classeur Excel. Ces applications sont appelées macros. La principale différence entre Visual Basic et Visual Basic pour Excel réside dans les collections d'objets : les objets gérés par Visual Basic sont différents de ceux de Visual Basic pour Excel.

Objets propres à Microsoft Excel

En Visual Basic, les données nécessaires pour effectuer certains calculs sont obtenues à l'aide d'un formulaire (*Form*). Dans celui-ci, sont inclus différents objets interagissant entre eux et permettant à l'utilisateur de saisir les informations : champs de texte, boutons radio, cases à cocher, etc. (*TextBox*, *OptionBox*, *CheckBox*, etc.). En Excel, le principe est le même : au lieu de saisir les informations à l'intérieur de *TextBox*, il s'agit d'utiliser les cellules incluses dans une feuille de calcul.

Exemple simple : Je veux obtenir 2 nombres et afficher le résultat de la somme.

En VB, le code pourrait ressembler à celui-ci.

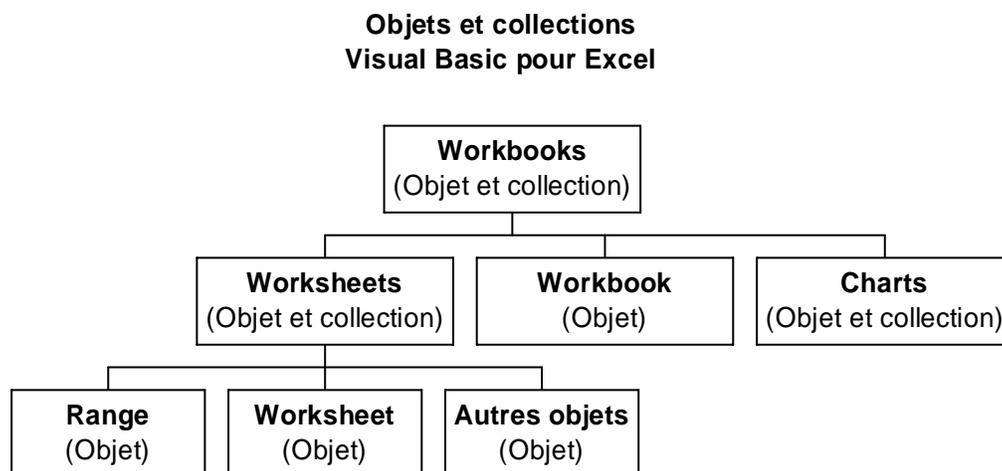
```
Formulaire.Reponse.Text = Val(Formulaire.Nombre1.Text) +  
Val(Formulaire.Nombre2.Text)
```

En VBA, le code ressemblerait à

```
Range("A1").Value = Range("A2").Value + Range("A3").Value
```

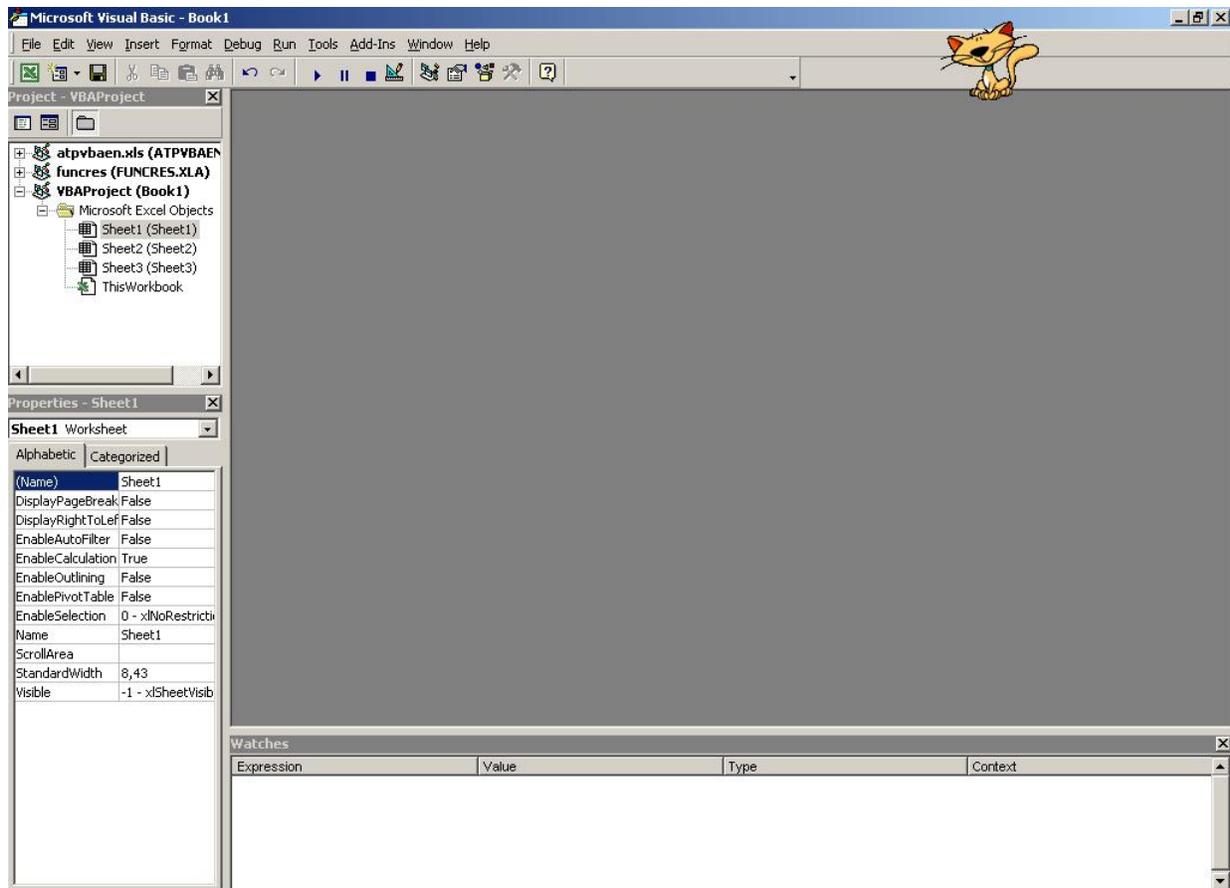
Note : Puisque la plupart du temps en Excel nous manipulons des chiffres, il n'est pas nécessaire d'utiliser une commande comme `Val` ou `Cdbl` pour convertir du texte en nombre.

Voici une vue d'ensemble des objets et collections les plus couramment utilisés en Visual Basic pour Excel. N'oubliez pas qu'une collection est un ensemble d'objets ou d'autres collections pouvant avoir certaines propriétés et pouvant aussi interagir avec d'autres éléments.



Bâtir une macro simple

À l'intérieur de Microsoft Excel, un petit sous-programme qui interagit avec les objets d'Excel est appelé une macro. Pour créer une macro par soi-même, il suffit d'aller dans le menu Outil, Macro, Éditeur Visual Basic.



Pour débiter la programmation, il suffit de sélectionner la feuille Excel dans laquelle la routine sera composée et d'aller dans le menu Affichage, Code.

Nous voulons reproduire le même tableau que dans la section précédente. Pour ce faire, nous pouvons créer une macro Excel qui fera ce boulot.

	A	B	C	D
1		Prestation de décès	10 000,00 \$	
2				
3		x	Prime assurance temporaire un an émis à l'individu d'âge x	
4	35	0,015	150,00 \$	
5	36	0,020	200,00 \$	
6	37	0,025	250,00 \$	
7	38	0,030	300,00 \$	
8	39	0,035	350,00 \$	
9	40	0,040	400,00 \$	
10				

Débutons la programmation avec la création de la routine `CreerTableau`.

```
Sub CreerTableau()
```

```
End Sub
```

Nous allons construire ce qui est inclus dans les cellules B1:C1. La construction du contenu des autres cellules sera expliquée un peu plus loin.

```
Sub CreerTableau()  
    'Ces deux lignes servent à saisir l'information dans les cellules  
    Range("B1").Value = "Prestation de décès"  
    Range("C1").Value = 10000  
  
    'Ces lignes servent à changer l'apparence du contenu des cellules  
    Range("B1:C1").Font.Name = "Arial"  
    Range("B1:C1").Font.Size = 12  
    Range("B1:C1").Interior.ColorIndex = 34  
    Range("B1:C1").Borders(xlEdgeTop).Weight = xlMedium  
    Range("B1:C1").Borders(xlEdgeBottom).Weight = xlMedium  
    Range("B1:C1").Borders(xlEdgeLeft).Weight = xlMedium  
    Range("B1:C1").Borders(xlEdgeRight).Weight = xlMedium  
    Range("B1").Font.Bold = True  
    Range("C1").Style = "Currency"  
  
    'Ces lignes servent à modifier la largeur des colonnes  
    Columns("B").ColumnWidth = 25  
    Columns("C").ColumnWidth = 15  
End Sub
```

Il est important de comprendre le fonctionnement de l'objet *Range*. Premièrement, cet objet correspond à une plage de cellules ou à une cellule. Cet objet a certaines propriétés (*Value*, *Style*) et contient aussi certains objets (*Font*, *Interior*, *Borders*). Ces derniers ont leurs propres propriétés ou attributs (*Name*, *Size*, *Interior*, *Weight*). Remarquez aussi la hiérarchie entre les différents objets. Par exemple, la propriété *Name* appartient à l'objet *Font* qui lui appartient à l'objet *Range*. Cette hiérarchie est définie avec les points. En lisant de gauche à droite nous pouvons dire «L'objet *Range* contient l'objet *Font* qui lui a la propriété *Name* qui a été fixée à *Arial*».

Note : L'objet *Font* n'appartient pas qu'à l'objet *Range* ; il peut aussi servir par exemple, à l'objet *ChartTitle* qui est aussi un objet.

Gestion des cellules et des plages de cellules : objets et méthodes de base

Range

Syntaxe

```
Range("cellule")  
Range("plage")  
Range("cellule1, cellule2, ..., celluleN")  
Range("plage1, plage2, ..., plageN")
```

Retourne un objet de type *Range* correspondant à la plage de cellule ou cellule correspondante.

Exemple

```
Range("A1")      `Stocke en mémoire le contenu de la cellule A1 et ses propriétés  
Range("A1:B6")   `Stocke en mémoire le contenu de la plage A1:B6 et ses propriétés  
Range("A1,C4,D6") `Stocke en mémoire le contenu des cellules A1,C4,D6, etc.  
Range("A1:B6, C3:D4") `Stocke en mémoire le contenu des plages A1:B6 et C4:D4,  
etc.
```

Columns ou Rows

Syntaxe

```
Columns("lettre de colonne")  
Columns(# de colonne)  
Rows(# de ligne)
```

Retourne un objet de type *Range* correspondant à la colonne spécifiée ou à la ligne spécifiée.

Exemple

```
Columns("C")     `Stocke en mémoire le contenu de la colonne C et ses propriétés  
Columns(3)  
Row(10)  
Columns("C").Font.Bold = True `Appliquera le style gras à toutes les cellules de  
la colonne C.
```

Selection

Syntaxe

Selection

Retourne un objet de type *Range* correspondant à la cellule ou à la plage de cellules sélectionnée.

Exemple

```
Selection.Font.Bold = True    'Applique la mise en forme gras à la plage  
sélectionnée
```

ActiveCell

Syntaxe

ActiveCell

Retourne un objet de type *Range* correspondant à la cellule active. Par exemple, si la cellule A1 est active, la saisie d'information au clavier se fera alors pour la cellule A1.

Exemple

```
ActiveCell.Font.Bold = True    'Applique la mise en forme gras à la cellule  
sélectionnée
```

Offset

Syntaxe

ObjetRange.Offset(position relative en ligne, position relative en colonne)

Retourne un objet de type *Range* correspondant à la cellule positionnée de façon relative à *ObjetRange*.

Exemple

```
Range("C8").Offset(-1,-2).Value équivaut à Range("A7").Value
```

Cells

Syntaxe

```
ObjetRangeOUWorksheet.Cells(position en ligne, position en colonne)  
ObjetRangeOUWorksheet.Cells(ième cellule)
```

Retourne un objet de type *Range* correspondant à la position fournie en argument.

Exemple

```
Range("B1:D4").Cells(2,2).Value équivaut à Range("C2").Value  
Worksheets("Feuille").Cells(1,3).Value équivaut à Range("C1").Value  
Range("B1:D4").Cells(4).Value équivaut à Range("B2").Value
```

Activate

Syntaxe

```
ObjetRangeCellule.Activate
```

Méthode qui permet d'activer une cellule. Par exemple, si la cellule A1 est active, la saisie d'information au clavier se fera alors pour la cellule A1.

Exemple

```
Range("A1").Activate
```

Select

Syntaxe

```
ObjetRange.Select
```

Méthode qui permet de sélectionner une cellule ou une plage de cellule.

Exemple

```
Range("A1:A15,C2:C26").Select
```

Remarque

Pour obtenir le contenu (valeurs ou textes) compris dans un certain objet *Range* et le stocker à l'intérieur d'une variable, il faut avoir un objet *Range* comprenant QU'UNE seule cellule.

Exemple (les prochaines lignes sont correctes)

```
Dim variable as Variant
variable = ActiveCell.Value
variable = Range("A1").Value
variable = Selection.Value ' Fonctionne si seulement UNE cellule est sélectionnée
variable = Range("C8").Offset(-1,-2).Value
```

Exemple (les deux dernières sont *incorrectes*)

```
Dim scalaire as Variant
Dim tableau(1 to 4,1 to 4) as Variant
scalaire = Range("A1:D4").Value
tableau = Range("A1:D4").Value
```

- La propriété *Value* ne s'applique que si l'objet *Range* correspond à UNE cellule.
- Il est quand même possible d'obtenir les valeurs contenues dans une plage de cellules avec la propriété *Cells*. Il faut par contre itérer sur toutes les lignes et les colonnes de la plage.
- Dans le cas où l'objet *Range* correspond à UNE cellule, il n'est pas nécessaire d'utiliser la propriété *Value* pour obtenir le contenu de la cellule.

Truc pour la manipulation répétitive du même objet

Lorsqu'on doit travailler souvent avec le même objet, il peut devenir très répétitif de toujours écrire la source de celui-ci. Par exemple, nous avons dû répéter à plusieurs reprises `Range("B1:C1")`. On peut par contre utiliser la structure `With... End With` qui permet de fixer l'utilisation d'un objet. Par exemple, la routine que nous avons conçue plus tôt aurait pu être écrite de cette façon.

```
Sub CreerTableau()

    'Ces deux lignes servent à saisir l'information dans les cellules
    Range("B1").Value = "Prestation de décès"
    Range("C1").Value = 10000

    'Ces lignes servent à changer l'apparence du contenu des cellules
    With Range("B1:C1")
        .Font.Name = "Arial"
        .Font.Size = 12
        .Interior.ColorIndex = 34
        .Borders(xlEdgeTop).Weight = xlMedium
        .Borders(xlEdgeBottom).Weight = xlMedium
        .Borders(xlEdgeLeft).Weight = xlMedium
        .Borders(xlEdgeRight).Weight = xlMedium
    End With
    Range("B1").Font.Bold = True
    Range("C1").Style = "Currency"

    'Ces lignes servent à modifier la largeur des colonnes
    Columns("B").ColumnWidth = 25
    Columns("C").ColumnWidth = 15

End Sub
```

Nous poursuivrons l'exemple du tableau après avoir regardé la programmation conditionnelle et les itérations.

Gestion des feuilles de calculs et des classeurs : objets et méthodes de base

Afin de mener à bon port la construction d'une macro quelconque, il est souvent essentiel de bien savoir manipuler les classeurs ainsi que les diverses feuilles les composant. À la fin de cette section, vous serez capable d'ouvrir des classeurs, de les enregistrer et de les fermer à l'aide des commandes Visual Basic pour Excel.

Tout d'abord, regardons les diverses commandes à exécuter pour manipuler des objets de type classeur (*Workbook*) et feuilles de calcul (*Worksheet*).

Workbooks

Syntaxe

```
Workbooks("nom du classeur")  
Workbooks(# du classeur)
```

Retourne un objet de type *Workbook* (classeur) correspondant au nom de classeur spécifié. Avec un objet *Workbook* en mémoire, il est ainsi possible d'exécuter certaines méthodes applicables aux classeurs tels que son ouverture, sauvegarde, fermeture, etc.

Remarque

La spécification du nom du classeur n'est pas obligatoire (par exemple si vous voulez utiliser certaines méthodes relatives aux classeurs en général, vous verrez plus loin). Par contre, si vous devez manipuler un classeur spécifique, l'argument devient obligatoire.

Exemple

```
Workbooks("Classeur1")
```

'Retourne un objet de type *Workbook* correspondant au classeur *Classeur1*. Vous pourrez donc manipuler le classeur *Classeur1* (fermer, sauvegarder, etc.)

```
Workbooks(1)
```

'Même chose, mais correspond au premier classeur ouvert.

ActiveWorkbook

Syntaxe

ActiveWorkbook

Retourne un objet de type *Workbook* correspondant au classeur présentement actif (sélectionné).

Exemple

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

Worksheets ou Sheets

Syntaxe

```
Worksheets("nom de la feuille de calcul")  
Worksheets(# de la feuille de calcul)  
Sheets("nom de la feuille de calcul")  
Sheets(# de la feuille de calcul)
```

Retourne un objet de type *Worksheet* (feuille de calcul) correspondant au nom de la feuille de calcul spécifié. Avec un objet *Worksheet* en mémoire, il est ainsi possible d'exécuter certaines méthodes applicables aux feuilles de calculs.

Remarque

La spécification du nom de la feuille de calcul n'est pas obligatoire (par exemple si vous voulez utiliser certaines méthodes relatives aux feuilles en général). Par contre, si vous devez manipuler une feuille de calcul spécifique, l'argument devient obligatoire.

Exemple

```
Sheets("Feuill")  
'Retourne un objet de type Worksheet correspondant à la feuille de calcul Feuill
```

```
Worksheets("Donnees").Range("A1").Value  
'Permet d'obtenir le contenu de la cellule A1. La cellule A1 est un objet de type Range. Un objet de type Range appartient à un objet de type Worksheet (Worksheets("Donnees") permet d'obtenir un objet de type Worksheet correspondant à la feuille Donnees). Puis, Value est une propriété associée à un objet de type Range (Range("A1") permet d'obtenir un objet Range correspondant à la cellule A1).
```

ActiveSheet

Syntaxe

`ActiveSheet`

Retourne un objet de type *Worksheet* correspondant à la feuille de calcul présentement active (sélectionnée).

Exemple

```
ActiveSheet.Cells(1,1).Value  
'Retourne le contenu de la cellule A1 de la feuille active.
```

Ouverture d'un classeur Excel – Méthode Open

Il est parfois très utile de pouvoir ouvrir un classeur *Excel* afin de pouvoir lire l'information contenue dans ce classeur.

Syntaxe

```
ObjetWorkbook.Open Filename:="chemin complet du fichier Excel"
```

Exemple

Voici la commande qui vous permettra d'ouvrir un classeur *Excel* :

```
Workbooks.Open Filename:="C:\Mes documents\Classeur.xls"
```

La commande ci-haut vous permettra d'ouvrir le classeur nommé *Classeur* dans le dossier *Mes documents* sur le disque *C*.

Création d'un nouveau classeur Excel – Méthode Add

Voici la commande qui vous permettra d'ouvrir un nouveau classeur *Excel* ne contenant aucune information :

Syntaxe

```
Workbooks.Add
```

Sélection (ou activation) d'un classeur ou d'une feuille de calcul – Méthodes Activate et Select

Afin de pouvoir accéder à l'information contenue dans une feuille ou dans un classeur donné, il est essentiel de sélectionner la feuille ou le classeur dans lequel les commandes exécutées s'appliqueront.

Syntaxe - Activation d'un classeur

```
ObjetWorkbook.Activate
```

Exemple

```
Workbooks("Classeur1").Activate
```

Syntaxe - Activation et sélection d'une feuille de travail

```
ObjetWorksheet.Activate
```

```
ObjetWorksheet.Select
```

Exemple

```
Sheets("feuille").Select
```

Sauvegarde d'un classeur *Excel* – Méthodes *SaveAs* et *Save*

Après avoir modifier l'information contenue dans un classeur, il peut être utile de vouloir sauvegarder cette information.

Syntaxe - Sauvegarder un nouveau fichier

```
ObjetWorkbook.SaveAs Filename:="nom complet du nouveau fichier Excel"
```

Exemple

```
ActiveWorkbook.SaveAs Filename:="C:\Mes documents\classeur1.xls"
```

Cette méthode vous permet de sauvegarder le classeur actif sous le nom *classeur1.xls* dans le dossier *Mes documents* sur le disque *C*.

Syntaxe - Sauvegarder un fichier existant

```
ObjetWorkbook.Save
```

Exemple

```
Workbooks("Classeur1").Save
```

Fermeture d'un classeur *Excel* – Méthode *Close*

Fermer les classeurs non utilisés est une bonne habitude de programmation puisqu'elle permet de réduire le nombre de fichiers actifs.

Syntaxe

```
ObjetWorkbook.Close
```

Exemple

Voici la commande qui vous permettra de fermer un classeur dont l'utilisation n'est plus requise.

```
Workbooks("Classeur1").Close
```

Déclaration de variables

Rappel provenant des sections précédentes

Variable : Espace en mémoire réservé pour y stocker de l'information. Une variable peut être de plusieurs types : numérique (entier, réel), booléen (ou logique), texte (ou chaîne de caractères, aussi appelé *string*).

Dans l'utilisation des variables, il y a toujours deux étapes nécessaires. Premièrement, la déclaration qui alloue ou réserve un espace en mémoire. Deuxièmement, le stockage utilise l'espace qui lui est réservé pour conserver l'information voulue.

Déclaration de variables scalaires

Une variable scalaire est une variable comprenant qu'UNE seule information. Cette information peut être de plusieurs types.

Syntaxe

Variables dont le contenu est conservé à l'intérieur d'une procédure ou d'une fonction (portée de niveau procédure)

```
Dim scalaire as Type
```

```
Dim scalaire1, scalaire2, ..., scalaireN as Type
```

```
Dim scalaire1 as Type1, scalaire2 as Type2, ..., scalaireN as TypeN
```

Variables dont le contenu est conservé tant et aussi longtemps que le programme principal n'est pas interrompu (portée de niveau module)

```
Private scalaire as Type
```

```
Private scalaire1, scalaire2, ..., scalaireN as Type
```

```
Private scalaire1 as Type1, scalaire2 as Type2, ..., scalaireN as TypeN
```

Remarque

Les variables à portée de niveau module doivent être déclarées avant toutes les procédures du module.

Types de données

(Tableau tiré du livre Excel 2000 et Visual Basic pour Applications 6)

<i>Types de données</i>	<i>Valeurs acceptées</i>	<i>Mémoire occupée</i>
Byte	Nombre entier compris entre 0 et 255	1 octet
Integer	Nombre entier compris entre -32768 et 32768	2 octets
Long	Nombre entier compris entre -2147483648 et 2147483647	4 octets
Simple	Nombre à virgule flottante compris dans $\pm 3.403E38$	4 octets
Double	Nombre à virgule flottante compris dans $\pm 1.7977E308$	8 octets
Currency	Nombre à virgule fixe avec quinze chiffres pour la partie entière et quatre chiffres pour la partie décimales compris dans ± 922337203685477.5808	8 octets

Il existe aussi les types de données *String* (chaînes de caractères), *Date*, etc.

Exemple

```
Private salaire as Double 'Variable à portée de niveau module

Sub calculeBonus()
    Dim bonus as Double 'Variable à portée de niveau procédure
End Sub
```

Déclaration de variables matrices

Une variable matrice est une variable comprenant plusieurs séries d'informations. Elle peut être comparée à un tableau ou à une matrice. Par contre, en Visual Basic tout comme dans tout autre langage de programmation, la variable matrice peut avoir plus que deux dimensions.

Syntaxe

Variables à dimensions définies

```
Dim Tableau(bi1 to bs1,bi2 to bs2,..., biN to bsN) as Type
Private Tableau(bi1 to bs1,bi2 to bs2,..., biN to bsN) as Type
```

```
Dim Tableau(n1,n2,..., nN) as Type
Private Tableau(n1,n2,..., nN) as Type
```

Variables à dimensions indéfinies

```
Dim Tableau() as Type
Private Tableau() as Type
```

Légende

- $bi1$ → représente l'adresse de la borne inférieure de la 1^{ère} dimension du tableau.
- $bs1$ → représente l'adresse de la borne supérieure de la 1^{ère} dimension du tableau.
- $n1$ → représente le nombre d'éléments dans la 1^{ère} dimension du tableau.

Remarques

- Les `bi1`, `bs1`, `ni1`, etc. doivent être des nombres entiers entrés par le programmeur. Ils ne peuvent représenter le contenu d'une variable.
- Si aucune adresse n'est définie, Visual Basic utilisera celle spécifiée par `Option Base`. Si la ligne `Option Base` n'est pas présente et que les adresses des bornes ne sont pas définies, les bornes iront de 0 jusqu'à `ni-1`.
- Si les dimensions d'une variable tableau ne sont pas définies, aucun contenu ne peut y être stocké tant et aussi longtemps que Visual Basic ne connaît pas ses dimensions. Vous pouvez redéfinir les dimensions d'une variable tableau à l'aide de la commande *ReDim* (voir plus loin).

Exemple # 1

À quoi pourraient servir les adresses des variables matrice ? Disons que vous simulez la performance d'un indice boursier entre 2003 et 2050. Déclarer la variable de cette façon...

```
Dim IndicesBoursiers(2003 to 2050) as Double
```

... et obtenir la simulation de l'année 2023...

```
SimulationVoulue = IndicesBoursiers(2023)
```

... est beaucoup plus simple que...

```
Dim IndicesBoursiers(48) as Double
```

```
SimulationVoulue = IndicesBoursiers(21)
```

Exemple # 2

```
Option Base 1
```

```
Sub NomRoutine()
```

```
    Dim Tableau4dimensions(10, 20, 30, 40) as Integer
```

```
End Sub
```

est équivalent à

```
Sub NomRoutine()
```

```
    Dim Tableau4dimensions(1 to 10, 1 to 20, 1 to 30, 1 to 40) as Integer
```

```
End Sub
```

Assigner et obtenir des valeurs d'une variable matrice

L'assignation d'une valeur à un élément d'une matrice se fait à peu près de la même façon qu'avec une variable scalaire (standard). Il faut évidemment spécifier la position dans la matrice où il faut stocker la donnée.

Exemple

```
Dim TableStandard(1 to 50, 1 to 25) as Double 'Déclaration de la variable matrice
TableStandard(2,3) = 1434 'Stocker la valeur 1434 dans la 2e ligne, 3e colonne de
la matrice Table standard.
```

```
Dim Tableau4dimensions(10, 20, 30, 40) as Integer 'Déclaration de la variable
matrice
```

```
Tableau4dimensions(1,1,1,1) = 1536 'Stocker la valeur 1536 à la position
(1,1,1,1) de la variable matrice.
```

L'affichage du contenu d'une variable matrice se fait de façon similaire.

Exemple (suite)

```
Range("C1").Value = TableStandard(2,3) 'Affiche dans la cellule C1 le contenu de  
la variable TableStandard à la ligne 2, colonne 3.
```

Redim

Nous avons mentionné plus tôt que les dimensions d'une variable matrice doivent être spécifiées explicitement (nombres entrés directement par le programmeur) ou pas spécifiées du tout. L'instruction *Redim* permet de redimensionner une variable matrice. Contrairement à l'instruction *Dim*, *Redim* peut accepter le contenu d'une variable.

Syntaxe

```
ReDim Tableau(bi1 to bs1,bi2 to bs2,..., biN to bsN) as Type  
ReDim Tableau(n1,n2,..., nN) as Type
```

Exemple

```
Dim nbresim as Integer  
Dim Simulations() as Double  
nbresim = 1000  
Redim Simulations(2003 to 2003 + nbresim - 1) as Double  
'car Dim Simulations(2003 to 2003 + nbresim - 1) as Double est illégal
```

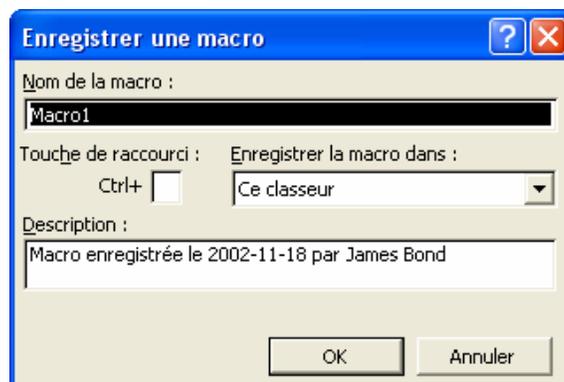
Création de routines et de fonctions personnelles

Il faut tout d'abord faire la distinction entre une routine et une fonction. Premièrement, une routine est un sous-programme qui exécute certaines commandes. Une fonction exécute aussi certaines commandes, mais en plus, retourne une certaine valeur ou tableau de valeurs.

Enregistrer une macro

Il existe plusieurs façons de créer des routines en VBA. L'une d'entre elles utilise la boîte de dialogue « Enregistrer une macro ». Cette méthode a l'avantage de ne nécessiter aucune programmation par l'utilisateur ; celui-ci n'a qu'à démarrer l'enregistrement de la macro, exécuter ce qui doit être emmagasiné dans la macro, et arrêter l'enregistrement. Cette façon de faire est utilisée lorsque la macro consiste en une procédure courante et répétitive que l'on a à faire, et que l'on désire automatiser. Voici comment procéder :

1. Dans le menu Outils de Excel, sélectionnez Macro, Nouvelle macro. La boîte de dialogue Enregistrez une macro s'affiche :



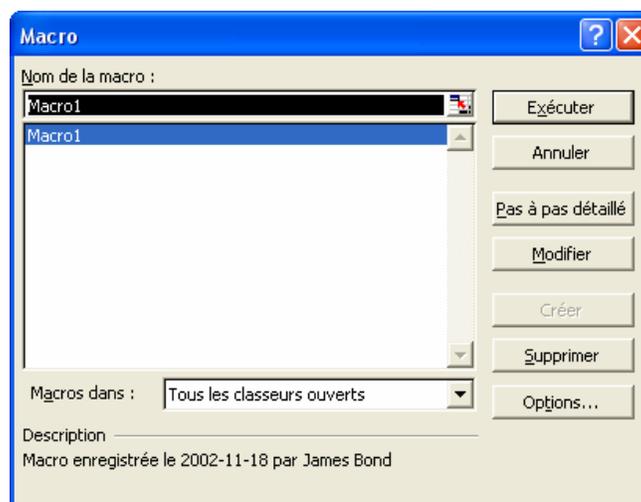
Il s'agit maintenant de donner un nom à la macro sous Nom de la macro. Il est possible d'attribuer une touche de raccourci à cette macro. Généralement, l'enregistrement de la macro se fait dans Ce classeur. Il est possible d'entrer une brève description de la macro.

2. Appuyez sur OK. La barre d'outils Arrêt de l'enregistrement s'affiche, indiquant que l'enregistrement de la macro est en cours.

Il ne reste plus qu'à exécuter (dans l'ordre) tout ce qui doit se retrouver dans la macro. Par exemple, quelqu'un pourrait écrire le chiffre 28 dans la cellule A1, changer le format de cellule en « monétaire » pour obtenir 28,00 \$ (bouton droit sur la cellule, Format de cellule, sélection de Monétaire sous l'onglet Nombre, ok), mettre la cellule en gras, pour ensuite arrêter l'enregistrement en appuyant sur le bouton carré de la barre d'outils Arrêt de l'enregistrement.

Une fois la macro enregistrée, il est possible de l'utiliser.

3. Effacez tout ce qui se trouve dans la cellule A1 (Édition, Effacer, tout).
4. Choisissez Macro, puis Macros... du menu Outils. La boîte de dialogue Macro s'affiche, vous permettant d'aller chercher la macro créée.



5. Appuyez sur Exécuter. La macro s'exécute, et refait ce que vous avez entré au préalable.

Il est possible d'aller voir la programmation qui a été nécessaire à la création de cette macro, et ce de deux façons. Vous pouvez retourner à la boîte de dialogue Macro (montrée ci-haut), et sélectionner Modifier. L'éditeur Visual Basic s'ouvrira et montrera les codes. Vous pouvez également tout simplement ouvrir Visual Basic en cliquant sur Macro, Visual Basic Editor du menu Outils.

```
Sub Macro1()  
,  
' Macro1 Macro  
' Macro enregistrée le 2002-11-18 par James Bond  
,  
  
,  
  
    Range("A1").Select  
    ActiveCell.FormulaR1C1 = "28"  
    Range("A1").Select  
    Selection.NumberFormat = "#,##0.00 $"  
    Selection.Font.Bold = True  
End Sub
```

Finalement, il est possible d'insérer un bouton dans la feuille Excel qui, lorsqu'on appuie dessus, exécute une macro.

1. Sous le menu Affichage, sélectionnez Barre d'outils, Formulaires. Une boîte à outils apparaît, que vous pouvez insérer dans la Barre d'outils et de menus en haut de l'écran.
2. Cliquez sur Bouton, représenté par un petit bouton gris. Votre curseur devient une croix. Il vous suffit maintenant de dessiner un bouton de la taille désirée.

Excel ouvre ensuite automatiquement la boîte de dialogue Affecter une macro. Vous pouvez maintenant sélectionner la macro que vous venez de créer et appuyer sur OK. Le bouton est maintenant fonctionnel, et exécutera la macro lorsque l'utilisateur appuie dessus. À noter qu'il est possible de changer le texte apparaissant sur le bouton, en cliquant dessus avec le bouton droit, puis le gauche. **Note importante** : Les macros que vous affectez à un bouton ne doivent pas recevoir d'arguments obligatoires.

Création de routines personnelles

Il a été mentionné plus tôt qu'il était préférable de diviser un problème en plusieurs petits problèmes. Le débogage s'en trouve ainsi simplifié. Une façon d'y parvenir est de créer une routine qui exécutera certaines commandes. Par exemple, une routine peut représenter une étape importante d'un algorithme.

Syntaxe

```
[Private ou Public] Sub NomDeLaRoutine(argument1 as Type1, argument2 as Type2,...)  
    Instructions  
End Sub
```

Remarques

- Un peu comme une variable, la routine peut avoir une portée privée ou publique, c'est-à-dire être disponible à un certain module Excel ou à tous les modules. Si le mot `private` ou `public` est omis, la routine sera publique.
- Il n'est pas nécessaire de spécifier des arguments à la routine. Vous pouvez aussi spécifier des arguments optionnels avec des valeurs par défaut. Voir la façon de procéder dans les fonctions personnelles, plus loin.
- Il est également possible d'assigner à un bouton une procédure créée de cette façon si la routine n'a aucun argument obligatoire.

Exemple

```
`Routine simple, de portée publique, sans arguments.  
Sub Programme()  
    Instructions  
End Sub
```

```
`Routine plus complexe, de portée privée, avec argument.  
Private Sub Initialisation(NomFichier as String)  
    Instructions  
End Sub
```

Appel de routines personnelles

Il y a deux façons d'appeler des routines personnelles dans Visual Basic pour Excel.

Syntaxe

```
`Première façon, Instruction Call  
  
Call NomDeLaRoutineSansArgument  
Call NomDeLaRoutineAvecArguments(Arg1, Arg2, ...)  
  
`Deuxième façon  
NomDeLaRoutineSansArgument  
NomDeLaRoutineAvecArguments Arg1, Arg2,...
```

Exemple

```
`Première façon, Instruction Call  
Call Programme  
Call Initialisation("autoexec.bat")  
  
`Deuxième façon  
Programme  
Initialisation "autoexec.bat"
```

Création de fonctions personnelles

Une autre façon de faciliter le débogage est l'utilisation de fonctions personnelles qui permet de vérifier certains calculs intermédiaires ou d'exécuter des calculs souvent exécutés dans le programme. Une fonction est différente d'une routine car elle retourne une valeur.

Syntaxe de la création de fonction

```
[Public ou Private] Function NomDeLaFonction(arguments) as Type  
    Instructions  
    NomDeLaFonction = expression  
End Function
```

Syntaxe des arguments (arguments)

```
NomArgument As Type  
[Optional] NomArgument As Type =(Valeur par défaut)
```

Remarques

- Un peu comme une variable, la fonction peut avoir une portée privée ou publique, c'est-à-dire être disponible à un certain module Excel ou à tous les modules. Si le mot `private` ou `public` est omis, la routine sera privée.
- Pour que la fonction puisse retourner une valeur, on doit absolument retrouver au moins une ligne de code ayant `NomDeLaFonction = expression`. Sinon la fonction retourne 0 et est inutile.
- La déclaration des arguments se fait la plupart du temps de la première façon, c'est-à-dire, `NomArgument As Type`.
- Un argument peut aussi être optionnel, c'est-à-dire que l'utilisateur pourrait appeler la fonction de façon correcte sans spécifier de valeur pour cet argument. Par contre, une valeur par défaut doit être présente. Une valeur par défaut est une valeur utilisée par un programme si l'utilisateur n'en spécifie aucune.
- Les arguments optionnels doivent être déclarés à la fin.
- La structure des arguments présentée dans cette section s'applique aussi aux routines.

Exemple

Voici la création d'une fonction avec un argument qui spécifie une valeur par défaut. Nous verrons comment appeler des fonctions avec des arguments par défaut et la différence. Plus bas, la même fonction est présentée, sans argument optionnel.

```
'Cet exemple, extrêmement simplifié de la réalité, calcule la cotisation à la RRQ.
'L'argument taux est optionnel
Function CotisationRRQavecTauxOpt(salaire As Double, Optional taux As Double =
0.035) As Double
    CotisationRRQavecTauxOpt = salaire * taux
End Function
```

```
'L'argument taux n'est PAS optionnel
Function CotisationRRQSansTauxOpt (salaire As Double, taux As Double) As Double
    CotisationRRQSansTauxOpt = salaire * taux
End Function
```

Appel de fonctions personnelles

Lorsque nous avons besoin du résultat d'un calcul complexe qui est heureusement, exécuté à l'intérieur d'une fonction personnelle, nous utiliserons la façon suivante pour appeler une fonction.

Exemple

Je voudrais calculer ma cotisation au régime des rentes du Québec avec mon salaire.

```
Dim MonSalaire, MaCotisation as Double
MonSalaire = 45000

'Appel de la fonction avec taux OPTIONNEL
MaCotisation = CotisationRRQavecTauxOpt(MonSalaire)
'Il y a 1575 dans la variable MaCotisation.

'Appel de la fonction avec taux OPTIONNEL
```

```
MaCotisation = CotisationRRQavecTauxOpt(MonSalaire, 0.04)
'Il y a 1800 dans la variable MaCotisation.
```

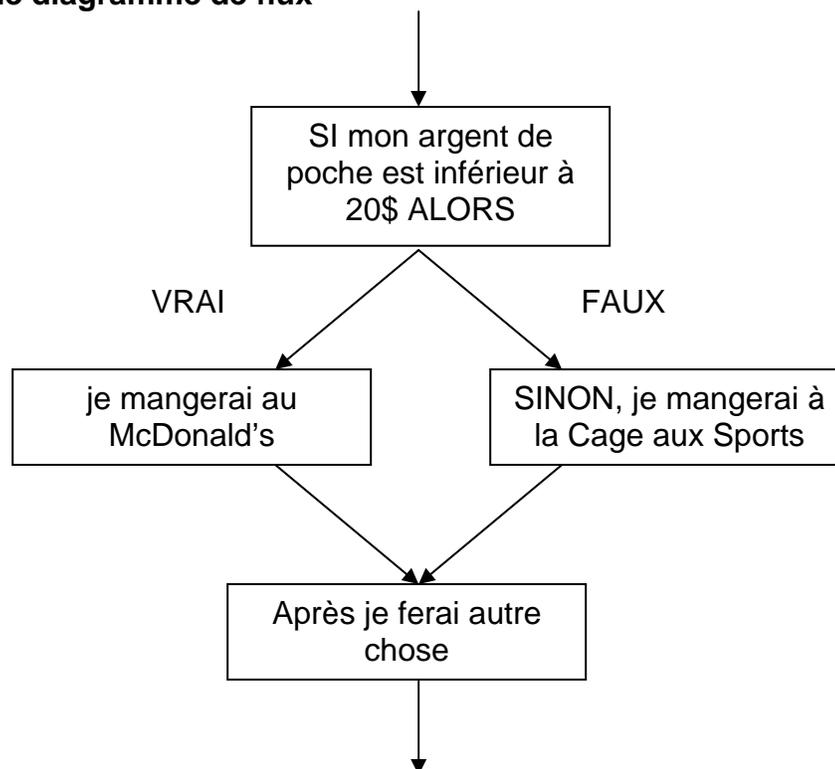
```
'Appel de la fonction SANS taux OPTIONNEL
MaCotisation = CotisationRRQSansTauxOpt(MonSalaire, 0.04)
'Il y aura toujours 1800 dans la variable MaCotisation.
```

```
'Appel de la fonction SANS taux OPTIONNEL
MaCotisation = CotisationRRQSansTauxOpt(MonSalaire)
'Retourne une erreur car le second argument n'est pas optionnel.
```

Programmation conditionnelle

La programmation conditionnelle permet d'exécuter certaines commandes selon la réalisation de certains événements ou l'obtention de certains résultats. Pour mieux comprendre les différentes possibilités, le programmeur peut utiliser un diagramme de flux. Vous remarquerez que la programmation conditionnelle à l'intérieur de Visual Basic pour applications est la même qu'à l'intérieur de Visual Basic.

Exemple de diagramme de flux



Toute programmation conditionnelle nécessite la construction d'une condition. Que ce soit en Visual Basic ou tout autre langage, une condition est formée par une ou plusieurs comparaisons. On forme une comparaison à l'aide des opérateurs de comparaison et on relie les comparaisons avec les opérateurs logiques.

Créer une condition

Avant tout, définissons les opérateurs de Visual Basic pour applications.

Opérateurs de comparaison

<	Strictement inférieur
>	Strictement supérieur
=	Est exactement égal à
<=	Inférieur ou égal
>=	Supérieur ou égal
<>	Est différent de

Opérateurs logiques

Les opérateurs logiques servent à unir les comparaisons pour former une condition.

And	ET (intersection)
Or	OU (union)

Notons tout d'abord `OPcomp` pour opérateur de comparaison et `OPlog` pour opérateur logique.

Syntaxe

`Element1 OPcomp Element2 OPlog Element3 OPcomp Element4 OPlog..`

Remarque

L'utilisation de parenthèses permet de déterminer un certain ordre entre les conditions.

Exemple

```
'Ce programme détermine si un étudiant va être engagé dans la firme d'actuaire ABC
Dim nbreexams as Integer
Dim moyenne as Double
Dim cyclesuperieur as Boolean
nbreexams = 3
moyenne = 3.23
cyclesuperieur = False

'Exemples de condition pour engager un étudiant (n'est pas basé sur la réalité...
exemple fictif)
'L'employeur veut que l'étudiant ait au moins 2 examens professionnels ET une
moyenne cumulative d'au moins 3. La condition est donc :
'nbreexams >= 2 And moyenne >= 3 'Retourne vrai car VRAI ET VRAI = VRAI
'L'employeur considère qu'un diplôme de cycle supérieur équivaut aux deux examens.
La condition est donc :
'(nbreexams >= 2 Or cyclesuperieur = True) And moyenne >= 3 'Retourne vrai car
(VRAI OU FAUX = VRAI) ET VRAI = VRAI

nbreexams = 1
```

```
cyclesuperieur = True  
moyenne = 3.51
```

'L'employeur veut que l'étudiant ait au moins 2 examens professionnels ET une moyenne cumulative d'au moins 3. La condition est donc :

```
'nbreexams >= 2 And moyenne >= 3 'Retourne faux car FAUX ET VRAI = FAUX
```

'L'employeur considère qu'un diplôme de cycle supérieur équivaut aux deux examens.

La condition est donc :

```
'(nbreexams >= 2 Or cyclesuperieur = True) And moyenne >= 3 'Retourne vrai car  
(FAUX OU VRAI = VRAI) ET VRAI = VRAI
```

Programmation conditionnelle simple

Syntaxe

```
If condition Then  
    commandes si condition est vraie (1)  
Else  
    commandes si condition est fausse  
End If
```

Remarques

- Peu importe la complexité de la condition, celle-ci doit être vraie pour exécuter les commandes en (1).
- Il est aussi possible d'imbriquer plusieurs blocs `If`.

Syntaxe

```
If condition1 Then  
    commandes si condition1 est vraie  
        If condition2 Then  
            commandes si condition1 ET condition2 sont vraies  
        Else  
            commandes si condition1 est vraie ET condition2 est fausse  
        End If  
Else  
    commandes si condition1 est fausse  
        If condition3 Then  
            commandes si condition1 est fausse ET condition3 est vraie  
        Else  
            commandes si condition1 ET condition3 sont fausses  
        End If  
End If
```

- On peut aussi laisser tomber le `Else` dans le cas où le programme ne devrait rien faire si la condition n'est pas respectée.

Programmation conditionnelle par cas

Syntaxe

```
Select Case variable
    Case condition1partielle
        commandes si condition1partielle est vraie
    Case condition2partielle
        commandes si condition2partielle est vraie
    ...
    Case Else
        Commandes si aucune condition partielle n'a été respectée
End Select
```

Remarques

Une condition partielle peut être :

- nombre si la comparaison équivaut à variable = nombre
- nombre1, nombre2 si la comparaison équivaut à variable = nombre1 Or variable = nombre2
- nombre1 to nombreN si la comparaison équivaut à variable = nombre1 Or variable = nombre2 Or ... Or variable = nombreN
- Is OPcomp nombre si la comparaison équivaut à variable OPcomp nombre

Un exemple viendra éclaircir le tout.

(Exemple tiré de l'aide de Visual Basic)

```
Select Case performance
    Case 1 'si performance=1
        Bonus = Salaire * 0.1
    Case 2, 3 'si performance = 2 ou 3
        Bonus = Salaire * 0.09
    Case 4 To 6 'si performance = 4 ou 5 ou 6
        Bonus = Salaire * 0.07
    Case Is > 8 'si performance > 8
        Bonus = 100
    Case Else 'si performance = 7 ou autre valeur
        Bonus = 0
End Select
```

Programmation itérative

Il arrive très fréquemment en programmation que certaines instructions doivent être répétées constamment. Le nombre de répétitions peut soit être fixée d'avance par une variable ou constante, ou le nombre de répétitions peut être conditionnel à l'évolution d'une certaine condition. Dans le premier cas, on utilisera les itérations simples, dans le second, les itérations conditionnelles.

Itération simple (*For... Next*)

Lorsque le nombre d'itérations à exécuter est fixé à l'intérieur du programme, que ce soit par une constante ou une variable, il est préférable d'utiliser une forme d'itération simple.

Syntaxe

```
For compteur = borneinf to bornesup
    Instructions
Next compteur
```

Remarques

- L'itération illustrée dans la syntaxe produira $(bornesup - borneinf + 1)$ itérations.
- Pour répéter n fois une certaine instruction, la forme la plus commune d'itération simple est

```
For compteur = 1 to n
    Instructions
Next compteur
```
- Il est également possible d'utiliser un compteur avec un pas différent de 1. En effet, si on inscrit `For compteur = borneinf to bornesup Step s`, la variable `compteur` prendra comme valeurs : `borneinf`, `borneinf + s`, `borneinf + 2s`, etc. et l'itération arrêtera une fois que la borne supérieure sera rencontrée.

Itération conditionnelle (*While... Wend*)

Lorsque nous voulons qu'une série d'instructions s'exécute tant et aussi longtemps qu'une certaine condition est respectée (reste vraie), nous utiliserons l'itération conditionnelle.

Syntaxe

```
Instructions d'initialisation
While condition
    Instructions si condition est vraie
Wend
```

Remarques

- Les instructions d'initialisation sont celles qui permettent d'entrer dans la boucle et celles qui donnent une certaine information (tel qu'un compteur).
- Un des éléments formant la condition DOIT se mettre à jour à l'intérieur de l'itération.
- La condition est bâtie de la même façon qu'à l'aide de la programmation conditionnelle.
- Si vous voulez utiliser un compteur, vous devrez le créer (en déclarant une variable) et l'incrémenter vous-mêmes à l'intérieur de la boucle.
- Attention aux boucles à l'infini ! Assurez-vous qu'il est possible que la condition passe de vrai à faux un jour !

Gestion des boîtes de dialogue

Lorsque nous voudrions obtenir de l'information d'un utilisateur, nous utilisons une boîte de dialogue appelée *InputBox*. La boîte *InputBox* est utilisée quand l'information transmise peut être du texte, un nombre, etc. Toutefois, lorsque nous voulons communiquer de l'information à l'utilisateur ou obtenir une réponse courte (oui ou non), nous utiliserons une boîte de type *MsgBox*.

Boîte de dialogue *InputBox*

Au lieu d'utiliser des cellules dans une feuille Excel afin d'obtenir des données de l'utilisateur, il peut être plus convenable d'utiliser une boîte *InputBox*.

Syntaxe

```
Variable = InputBox("question posée","titre de la boite","valeur par défaut")
```

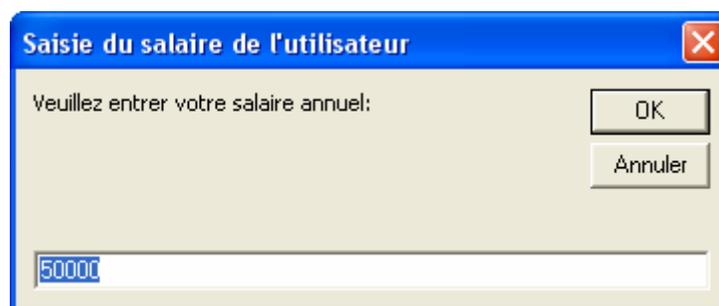
Arguments

"question posée"	est le texte qui apparaîtra dans la boîte de dialogue, qui est généralement une question posée à l'utilisateur.
"titre de la boite"	est facultatif et représente le titre de la boîte de dialogue; s'il est omis, le nom de l'application apparaîtra.
"valeur par défaut"	est lui aussi facultatif, correspond au texte apparaissant par défaut dans la zone de texte lors de l'affichage de la boîte de dialogue.

Exemple

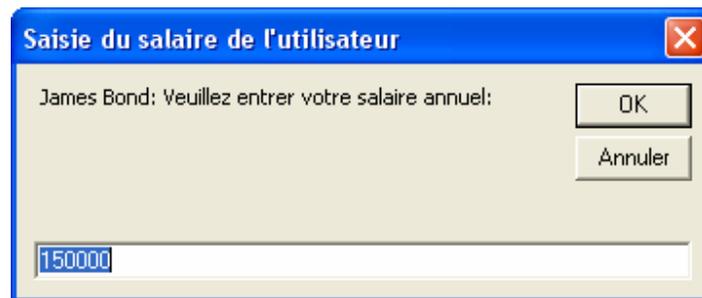
Admettons que l'on veuille obtenir le salaire annuel de l'utilisateur. Cette procédure stockerait le salaire obtenu de l'utilisateur dans la variable « salaire » :

```
Sub Salaire_annuel()  
  
Dim salaire As Single  
  
salaire = InputBox("Veuillez entrer votre salaire annuel:", _  
"Saisie du salaire de l'utilisateur", 50000)  
  
End Sub
```



À noter qu'il est possible d'insérer des variables dans la fonction *InputBox* avec l'opérateur de concaténation « & ». Par exemple, si le nom de l'utilisateur est inscrit dans la cellule A1, on pourrait avoir le programme suivant :

```
Sub Salaire_annuel()  
  
Dim salaire As Single  
Dim nom As String  
  
nom = Range("a1")  
salaire = InputBox(" " & nom & ": Veuillez entrer votre salaire annuel:", _  
"Saisie du salaire de l'utilisateur", 150000)  
  
End Sub
```



Boîte de dialogue *MsgBox*

Il peut être utile de fournir un renseignement à l'utilisateur pendant l'exécution d'un programme. Dans ce cas, l'utilisation de la boîte *MsgBox* est appropriée.

Syntaxe – Afficher une information

```
MsgBox "Message"  
MsgBox("Message", Icône, "Titre de la boîte")
```

Arguments

"Message"	est le texte qui apparaîtra dans la boîte de dialogue. Est en quelque sorte le message destiné à l'utilisateur
Icône	est facultatif et représente un nombre entier. Ce nombre correspond à l'icône choisie dans la boîte de dialogue.
"Titre de la boîte"	est lui aussi facultatif, correspond au titre de la boîte de dialogue.

Exemple

```
Sub Investigation()  
  
    MsgBox "Cet employé n'est plus rentable."  
  
End Sub
```



Il est également possible d'obtenir des informations de la part de l'utilisateur avec la boîte MsgBox, lorsqu'il s'agit de répondre à une question par l'affirmative ou la négative.

Syntaxe – Obtenir une information simple

```
Variable = MsgBox("Message", BoutonsEtIcône, "Titre de la boîte")
```

Arguments

"Message"	est le texte qui apparaîtra dans la boîte de dialogue. Est en quelque sorte le message destiné à l'utilisateur
BoutonsEtIcône	est facultatif et représente un nombre entier. Ce nombre correspond aux boutons et icônes qui apparaîtront dans la boîte de dialogue. Voir tableau plus bas.
"Titre de la boîte"	est lui aussi facultatif, correspond au titre de la boîte de dialogue.

Remarques

- La variable « Variable » est de type *Integer*.
- L'argument `BoutonsEtIcône` peut s'entrer de plusieurs façons. Voici un tableau qui les résume :

<i>Constante</i>	<i>Valeur</i>	<i>Description</i>
Bouton		
<code>vbOKOnly</code>	0	OK
<code>vbOKCancel</code>	1	OK et Annuler
<code>vbAbortRetryIgnore</code>	2	Abandonner, Réessayer et Ignorer
<code>vbYesNoCancel</code>	3	Oui, Non et Annuler
<code>vbYesNo</code>	4	Oui et Non
<code>vbRetryCancel</code>	5	Réessayer et Annuler
Symbole ou Icône		
<code>vbCritical</code>	16	Message critique
<code>vbQuestion</code>	32	Question
<code>vbExclamation</code>	48	Stop
<code>vbInformation</code>	64	Information
Bouton actif par défaut		
<code>vbDefaultButton1</code>	0	Premier bouton
<code>vbDefaultButton2</code>	256	Deuxième bouton
<code>vbDefaultButton3</code>	512	Troisième Bouton
<code>vbDefaultButton4</code>	768	Quatrième bouton

- Il est possible de combiner boutons et icônes en additionnant les valeurs de la colonne du centre. Voir exemple.
- Lorsque l'utilisateur appuie sur une touche (oui, non, annuler, etc.), la commande *MsgBox* retourne une valeur entière. Les valeurs sont résumées dans le tableau suivant.

<i>Bouton</i>	<i>Constante</i>	<i>Valeur</i>
OK	vbOK	1
Annuler	vbCancel	2
Abandonner	vbAbort	3
Réessayer	vbRetry	4
Ignorer	vbIgnore	5
Oui	vbYes	6
Non	vbNo	7

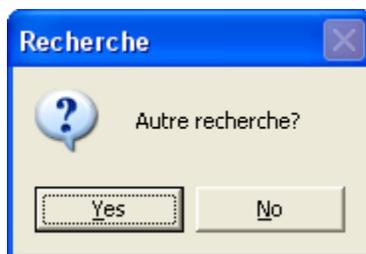
Ce tableau se lit de la façon suivante : si l'utilisateur appuie sur le bouton Oui, alors *MsgBox* retourne la valeur 6. La variable *variable* a donc le nombre 6 stocké.

Note : Ces deux tableaux sont tirés du livre Excel 2000 et Visual Basic pour Applications 6

Exemple

Si par exemple nous désirons afficher une boîte de dialogue contenant le symbole « Question », dans laquelle l'utilisateur pourra répondre à la question posée par « Oui » ou « Non », l'argument *BoutonsEtIcône* sera égal à 36 : 4 pour les boutons + 32 pour le symbole. Les trois instructions suivantes sont équivalentes et produisent la même boîte :

- `reponse = MsgBox("Autre recherche?", 36, "Recherche")`
- `reponse = MsgBox("Autre recherche?", 32 + 4, "Recherche")`
- `reponse = MsgBox("Autre recherche?", vbYesNo + vbQuestion, "Recherche")`

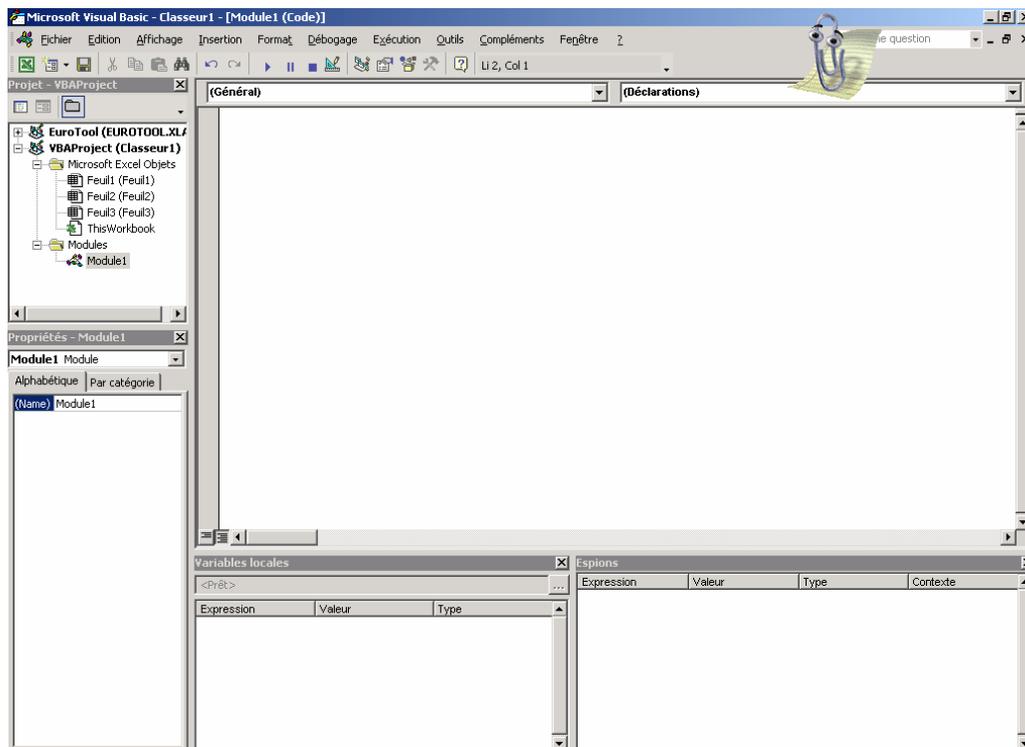


Dans les instructions précédentes, la variable *reponse* contient la réponse de l'utilisateur à la question. Cette réponse est en fait un chiffre; le programme interprétera ensuite ce chiffre et agira en conséquence, selon les désirs du programmeur.

Exemple tiré du livre Excel 2000 et Visual Basic pour Applications 6.

Une vue d'ensemble de l'éditeur Visual Basic

Visual Basic Editor est l'environnement de développement intégré de VBA. Voici un aperçu de Visual Basic Editor :

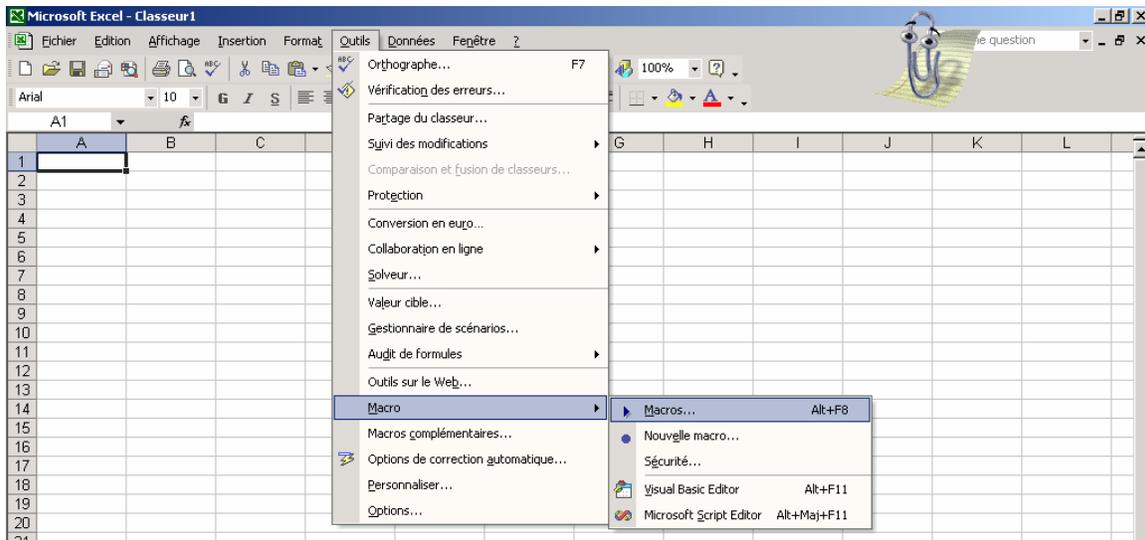


Intéressons-nous à quelques éléments qui pourront vous être utiles.

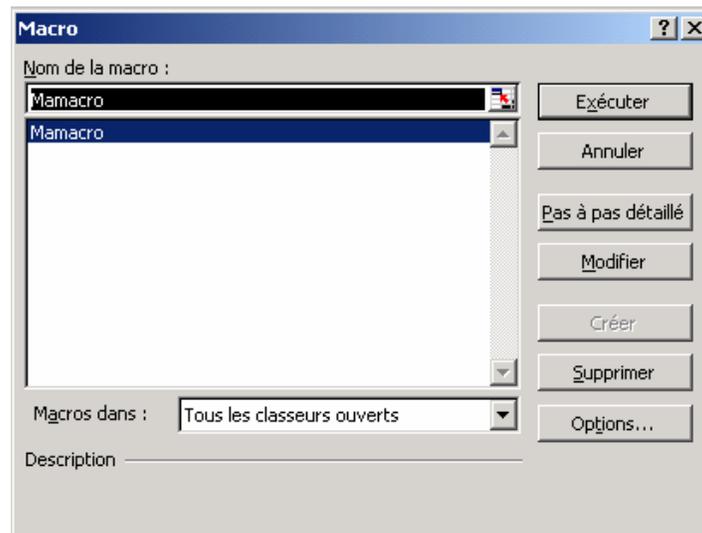
Trucs de débogage

Commande Exécuter

La commande *Exécuter* permet de mettre en marche la macro construite avec VBA. Il existe deux façons de procéder à l'exécution de la macro. La façon la plus rapide consiste à appuyer sur la touche  située dans le haut de la fenêtre de commande. Pour exécuter la macro sans accéder à l'éditeur VBA, on doit choisir Outils/Macro/Macros comme l'illustre la figure suivante :



Voici la boîte de dialogue qui apparaîtra :



Il ne suffit que d'enfoncer le bouton *Exécuter* pour exécuter la macro souhaitée.

Il est également possible d'exécuter une macro à l'aide de la touche F5.

Exécuter Pas à pas

Lorsqu'un programme ne produit pas le résultat escompté, sans générer d'erreurs, le premier test consiste à exécuter la procédure *pas à pas*, afin d'en examiner le déroulement et les conséquences sur le document, instruction après instruction.

Pour exécuter un programme VBA pas à pas, on doit adopter la procédure suivante :

- (1) Placer le curseur à la ligne où vous souhaitez démarrer l'exécution pas à pas dans la fenêtre de code de la procédure à tester;
- (2) Enfoncer la touche F8 pour exécuter la première ligne du code;
- (3) À chaque pression de la touche F8, la ligne de code suivante sera exécutée.

Si vous êtes convaincu de l'exactitude de votre programme jusqu'à une certaine ligne, positionnez votre curseur à cette ligne, appuyez sur CTRL+F8 (appuyez sur CTRL et maintenez enfoncé, puis appuyez sur F8). Visual Basic aura exécuté les lignes avant le curseur et est prêt à exécuter les prochaines lignes, pas-à-pas, avec vous.

Truc : lors de la lecture pas à pas d'une procédure, il peut être intéressant de voir ce que contiennent les variables au fur et à mesure qu'elles se remplissent. Lorsque vous exécutez une procédure en mode pas à pas, vous n'avez qu'à placer le curseur de la souris pour voir ce que contient une variable.

Points d'arrêt

Les points d'arrêt permettent d'interrompre l'exécution d'un programme sur une instruction précise. Cette possibilité est particulièrement intéressante lorsque vous soupçonnez l'origine d'une erreur. Vous pouvez ainsi exécuter normalement toutes les instructions ne posant pas de problèmes et définir un point d'arrêt pour une instruction dont vous n'êtes pas sûr.

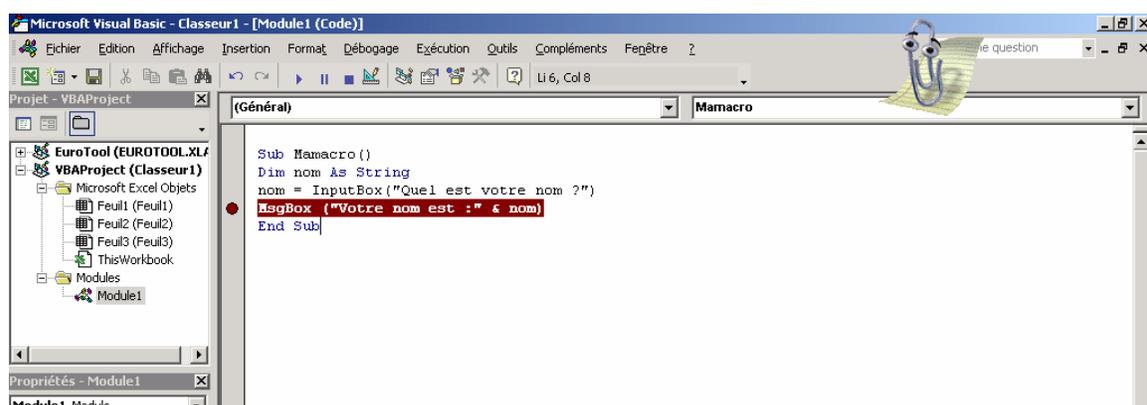
La procédure à adopter afin d'insérer un point d'arrêt est simple :

1. Placez le curseur sur l'instruction voulue;
2. Choisir la commande Basculer le point d'arrêt du menu Débogage ou appuyer sur la touche F9;
3. Pour supprimer le point d'arrêt, procéder de la même façon que pour insérer un point d'arrêt.

Autre technique

Il suffit de cliquer en marge de l'instruction à gauche (zone grise). L'instruction en question sera mise en évidence en rouge vin, et un rond rouge vin viendra se placer en marge. Pour enlever le point d'arrêt, il suffit de cliquer dessus à nouveau.

Voici une brève illustration :



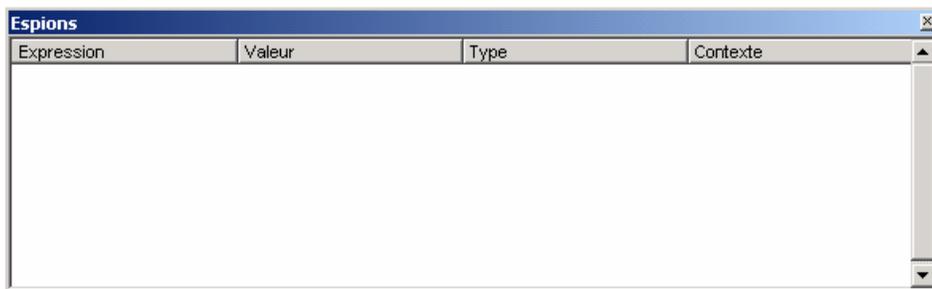
Fenêtre espions

Les espions permettent d'espionner les valeurs de variables ou de toute expression renvoyant une valeur dans un contexte déterminé.

Voici la procédure à adopter pour créer un espion :

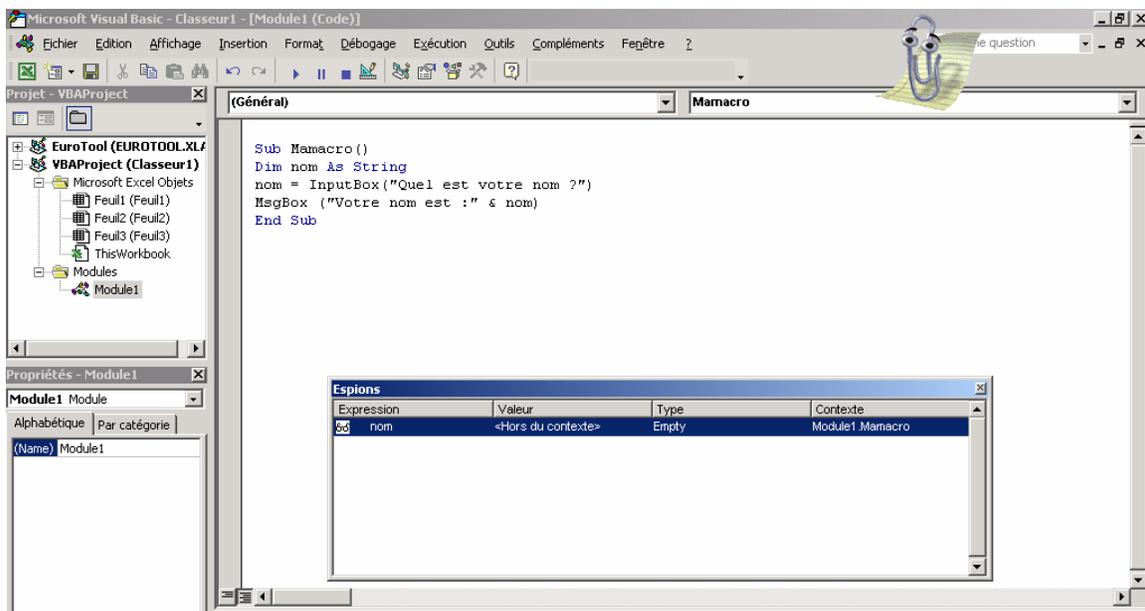
- (1) Choisissez la commande *Fenêtre espions* du menu *Affichage*;

La fenêtre suivante apparaîtra :



- (2) Mettre en surbrillance la variable à espionner dans la fenêtre de code, cliquer sur cette variable et la glisser dans la fenêtre espions en maintenant enfoncée la touche de gauche de la souris.

Dans l'exemple précédent, on obtiendrait le résultat suivant :



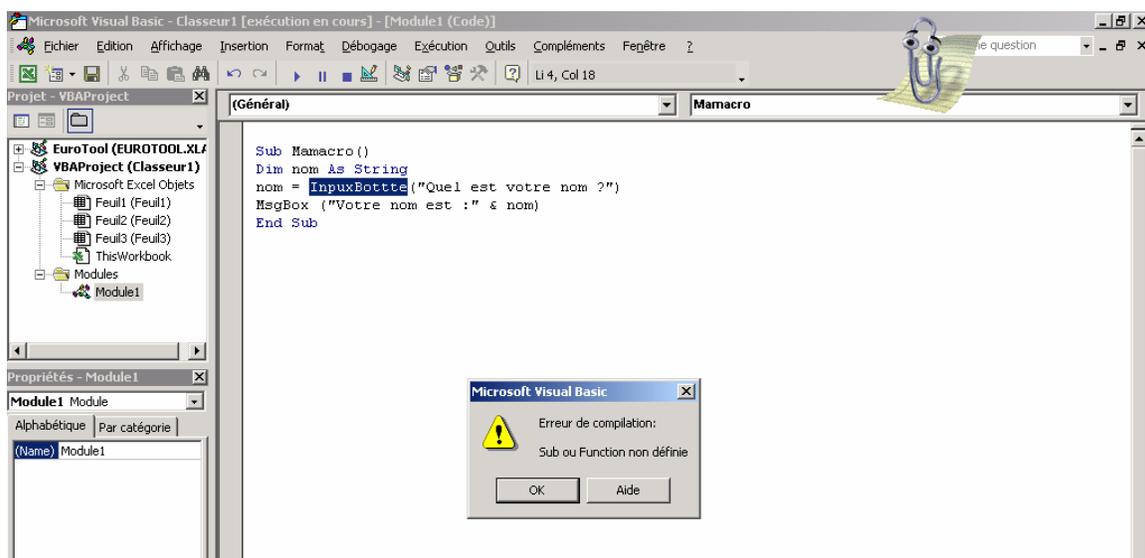
En exécutant le code en mode pas-à-pas ou à l'aide de points d'arrêt, vous serez en mesure de voir le contenu des variables (placées dans la fenêtre espion) se mettre à jour.

Messages d'erreurs

Le débogage consiste donc à régler les erreurs directement liées au code du programme et indépendantes de l'environnement dans lequel s'exécute le programme. Trois principaux types d'erreurs peuvent affecter un programme VBA soit les erreurs de compilation, les erreurs d'exécution et les erreurs logiques.

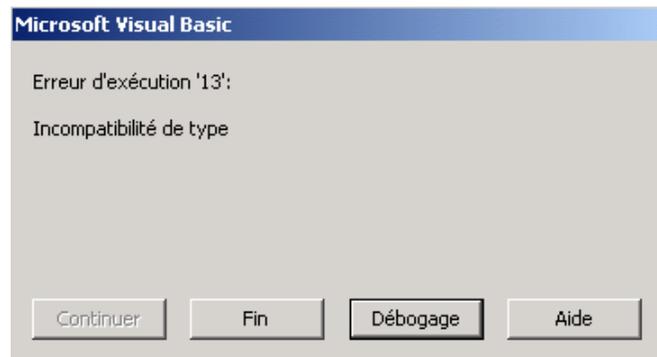
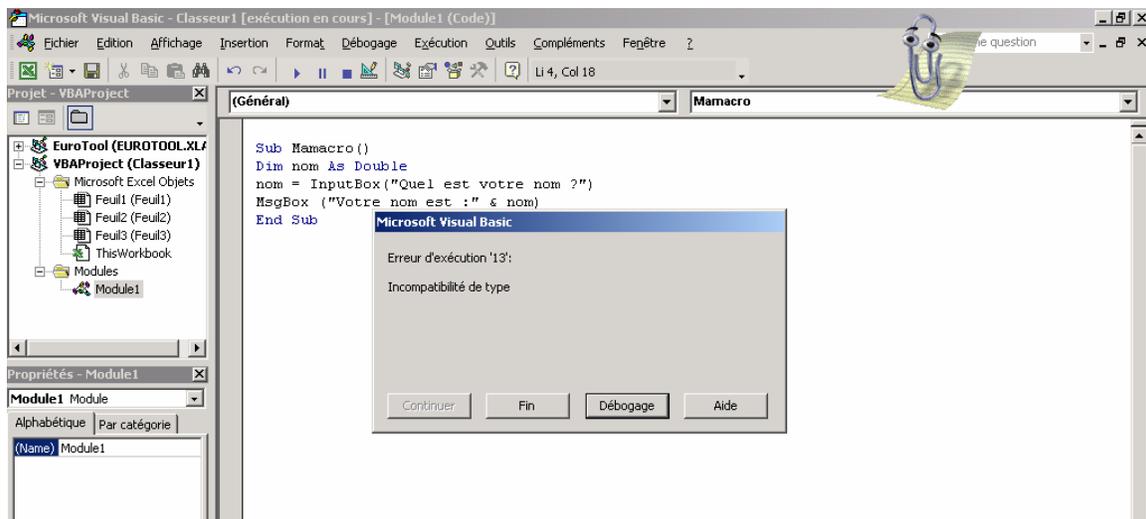
Erreurs de compilation

Ce type d'erreur survient lorsque VBA rencontre une instruction qu'il ne reconnaît pas; par exemple, lorsqu'un mot clé contient une faute d'orthographe. On remarque que ce type d'erreur est généralement le plus fréquent. Si on se réfère à l'exemple précédent, on aurait pu obtenir ce message d'erreur :



Erreurs d'exécution

Ce type d'erreur survient après que la compilation du programme a été effectuée avec succès. Une erreur d'exécution peut par exemple, être liée à l'utilisation de données incompatibles. Dans notre exemple, si on avait déclaré la variable *nom* comme un double, on aurait obtenu le message d'erreur suivant si on avait entré un nom en lettres lors de l'exécution du programme.



Dans la fenêtre ci-dessus le bouton *Fin* permet de terminer l'exécution du programme, le bouton *Débogage* met en surbrillance la ligne de code où VBA détecte une erreur et le bouton *Aide* affiche la rubrique d'aide associée à l'erreur reconnue.

Erreurs logiques

Ce type d'erreur est le plus ardu à corriger. Contrairement aux erreurs de compilation et d'exécution, elles laissent le programme s'exécuter. Le résultat obtenu ne sera pas celui que vous escomptiez. Les erreurs logiques découlent généralement d'une lacune de l'algorithme. Une combinaison de la méthode pas à pas et de l'utilisation des espions est une bonne méthode pour contrer ce type d'erreurs.

Suite de l'exemple – Construction du tableau de primes

La suite du code de l'exemple a été conçue de façon à pouvoir pratiquer le plus de concepts possibles. Il n'est pas nécessairement le plus efficace.

Les lignes qui suivent comprennent tout le code nécessaire pour reproduire l'illustration suivante.

	A	B	C	D
1		Prestation de décès	10 000,00 \$	
2				
3	x	Probabilité de décès dans la prochaine année d'un individu d'âge x	Prime assurance temporaire un an émis à l'individu d'âge x	
4	35	0,015	150,00 \$	
5	36	0,020	200,00 \$	
6	37	0,025	250,00 \$	
7	38	0,030	300,00 \$	
8	39	0,035	350,00 \$	
9	40	0,040	400,00 \$	
10				

'Avec ce programme, les cellules C4:C9 ne seront pas mises à jour si on change 'la prestation de décès. Il faudra exécuter le programme à chaque fois que la prestation 'change.

'Cette routine est fonction d'une prestation de décès optionnelle
Sub CreerTableau(Optional Prestation As Double = 10000)

```
'Ces deux lignes servent à saisir l'information dans les cellules
Range("B1").Value = "Prestation de décès"
Range("C1").Value = Prestation
```

```
'Ces lignes servent à changer l'apparence du contenu des cellules
With Range("B1:C1")
    .Font.Name = "Arial"
    .Font.Size = 12
    .Interior.ColorIndex = 34
    .Borders(xlEdgeTop).Weight = xlMedium
    .Borders(xlEdgeBottom).Weight = xlMedium
    .Borders(xlEdgeLeft).Weight = xlMedium
    .Borders(xlEdgeRight).Weight = xlMedium
End With
```

```
Range("B1").Font.Bold = True
Range("C1").Style = "Currency"
```

```
'Ces lignes servent à modifier la largeur des colonnes
Columns("B").ColumnWidth = 25
Columns("C").ColumnWidth = 15
```

```
'Ces lignes servent à entrer le texte
Range("A3").Value = "x"
Range("B3").Value = "Probabilité de décès dans la prochaine année d'un individu d'âge x"
Range("C3").Value = "Prime assurance temporaire un an émis à l'individu d'âge x"
```

```
Dim PlagePrimes As Range
Set PlagePrimes = Range("C4:C9")
For i = 0 To 5
    Cells(4 + i, 1) = 35 + i
    Cells(4 + i, 2) = 0.015 + 0.005 * i
    PlagePrimes.Cells(i + 1).Value = Prestation * Cells(4 + i, 2).Value
Next i

'Les prochaines lignes servent à formater le tableau (sans la bordure)

Dim PlageTitre As Range
Dim PlageContenu As Range

Set PlageTitre = Range("A3:C3")
Set PlageContenu = Range("A4:C9")

PlageTitre.Font.Name = Arial
PlageTitre.Font.Size = 12
PlageTitre.Font.Bold = True
PlageTitre.WrapText = True
PlageTitre.VerticalAlignment = xlVAlignCenter
PlageTitre.HorizontalAlignment = xlHAlignCenter
PlageTitre.Interior.ColorIndex = 34

With PlageContenu
    .Font.Name = Arial
    .Font.Size = 12
    .HorizontalAlignment = xlHAlignCenter
    .Interior.ColorIndex = 6
    .Columns(2).NumberFormat = "0.000"
    .Columns(3).Style = "currency"
End With

'Les prochaines lignes appliquent la bordure
With Range("A3:C9")
    .Borders(xlEdgeTop).Weight = xlThick
    .Borders(xlEdgeBottom).Weight = xlThick
    .Borders(xlEdgeLeft).Weight = xlThick
    .Borders(xlEdgeRight).Weight = xlThick
End With

With Range("A3:B9")
    .Borders(xlEdgeTop).Weight = xlThick
    .Borders(xlEdgeBottom).Weight = xlThick
    .Borders(xlEdgeLeft).Weight = xlThick
    .Borders(xlEdgeRight).Weight = xlThick
End With

With Range("A4:C9")
    .Borders(xlEdgeTop).Weight = xlThick
    .Borders(xlEdgeBottom).Weight = xlThick
    .Borders(xlEdgeLeft).Weight = xlThick
    .Borders(xlEdgeRight).Weight = xlThick
End With

Range("A3:A9").Borders(xlEdgeRight).Weight = xlThin

'Les valeurs xlThin, xlThick, etc. sont des nombres dont la valeur
```

```
'a déjà été stockée en mémoire dans ces constantes systèmes.  
End Sub  
  
'Cette routine permet d'appeler la routine CreerTableau avec une nouvelle  
'prestation de décès  
Sub PRINCIPAL()  
    CreerTableau 15000  
End Sub
```

Remarques

- Pour que les cellules C4:C9 soient mises à jour dès que l'on modifie la prestation de décès, on peut enlever la ligne
`PlagePrimes.Cells(i + 1).Value = Prestation * Cells(4 + i, 2).Value`
et utiliser la propriété *Formula* associé à un objet *Range*.

Exemple

```
Range("C4").Formula = "=B4*$C$1"
```

- Vous pouvez aussi mettre en pratique toutes vos connaissances de Visual Basic et Visual Basic pour Excel en faisant l'exercice actuariel # 5 et en parcourant rapidement les travaux pratiques de l'année 2000.

Éléments de programmation avancée en Visual Basic pour Excel

Objet *WorksheetFunction*

À l'intérieur de Visual Basic pour Excel, le programmeur a accès aux fonctions de Microsoft Excel afin qu'il puisse exécuter certains calculs de façon beaucoup plus rapide, c'est-à-dire sans qu'il ait à programmer d'autres routines faisant le même boulot.

Exemple

Un professeur a besoin de la moyenne des notes de ses étudiants à l'intérieur d'une de ses routines. Supposez que celles-ci sont stockées dans la variable `Notes(1 to 50) as Double`.

```
'Au lieu de faire..
```

```
Dim moyenne,somme as Double
```

```
For i = 1 to 50
```

```
    somme = somme + Notes(i)
```

```
Next i
```

```
moyenne = somme / 50
```

'Le professeur peut obtenir le même résultat à l'aide d'une toute petite ligne de code.

```
moyenne = WorksheetFunction.Average(Notes)
```

Syntaxe

```
VariableScalaire = WorksheetFunction.NomFonction(argument1,argument2,...)
```

Remarques

- Le nom de la fonction est toujours en anglais, peu importe la langue d'Excel.
- `VariableScalaire` doit être du même type que l'output de la fonction `NomFonction`.
- `Argument1`, etc. sont les arguments associés à `NomFonction`. Ceux-ci peuvent être le contenu de variables (de type correspondant) ou être des objets de type *Range* (dont le contenu est de type correspondant).

Fonctions les plus utilisées (`NomFonction`)

Fonctions mathématiques de base

- **SUM** → calcule la somme des éléments
- **AVERAGE** → calcule la moyenne des éléments
- **MDETERM** → calcule le déterminant d'une matrice

Fonctions statistiques

- **MIN** → retourne le minimum des éléments
- **MAX** → retourne le maximum des éléments
- **MEDIAN** → retourne la médiane des éléments
- **STDEV** → calcule l'écart-type des éléments (en supposant échantillon)

- STDEVP → calcule l'écart-type des éléments (en supposant population)
- CORREL → calcule le coefficient de corrélation entre deux séries de données
- SKEW → calcule le coefficient d'asymétrie (skewness) des éléments
- KURT → calcule le coefficient d'aplatissement (kurtosis) des éléments
- Toutes les distributions (NORMDIST, POISSON, TDIST, GAMMADIST, etc.)

Fonctions financières

Fonctions logiques

etc.

Nous vous invitons à consulter l'aide d'Excel pour une liste de toutes les fonctions et de leurs arguments.

Remarque

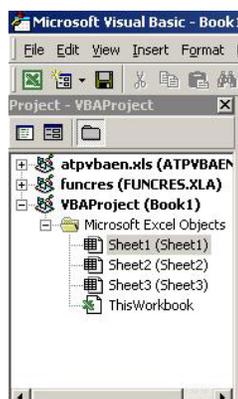
Lorsque certaines fonctions retournent des vecteurs ou des matrices de valeurs comme MMULT (multiplie deux matrices) et MINVERSE (inverse une matrice), l'utilisation de `WorksheetFunction` devient un peu plus corsée, mais toujours possible. Il faut remplacer `VariableScalaire` par une variable multi-dimension de type variable (*Variant*). Ceci dépend par contre de la version utilisée de Microsoft Excel.

Utiliser une fonction publique Visual Basic créée par le programmeur à l'intérieur d'un classeur

Il est possible de rendre disponible à l'intérieur d'une feuille de calcul toute fonction publique que vous avez créée préalablement. En effet, toutes les fonctions Excel pré installées comme SOMME, MOYENNE, etc. sont en fait que des fonctions publiques créées à l'intérieur de Visual Basic. Par contre, vous n'avez pas accès à ces fonctions pré installées.

Créer une fonction publique en 3 étapes simples

1. Ouvrir l'éditeur Visual Basic.
2. À gauche dans la zone Projet, sous Objets Microsoft Excel, cliquez sur Modules et ensuite allez dans le menu Affichage, Code. Si vous ne voyez pas Modules, cliquez sur Objets Microsoft Excel avec le bouton droit de la souris, puis cliquez sur Insertion, puis Modules.



3. Créer une fonction publique de la même façon que vous avez bâti toute fonction ou sous-routine.

Exemple complet

Dans Excel, il existe une panoplie de fonctions pré installées qui nous permettent d'obtenir la distribution de probabilité de plusieurs lois. Nous allons créer le même type de fonction, pour la loi uniforme continue.

Exécutez les deux premières étapes de la section précédente, puis utilisez le code ci-dessous. Cet exemple permet de bien maîtriser l'utilisation de l'objet `WorksheetFunction`, la programmation conditionnelle par cas ainsi que la structure `With ... End With`.

```
Public Function UniformContDist(x As Double, a As Double, b As Double, cumulative
As Boolean)
```

```
'a est la borne inférieure de la distribution uniforme
'b est la borne supérieure de la distribution uniforme
'cumulative est un argument de type boolean. Si true alors la fonction calculera
'la fonction de répartition à x. Sinon, la fonction de densité sera évaluée à x.
```

```
Select Case cumulative
    Case True
        With WorksheetFunction
            UniformContDist = .Min(.Max((x - a) / (b - a), 0), 1)
        End With
    Case False
        UniformContDist = 1 / (b - a)
End Select

End Function
```

Ensuite, retournez dans une feuille de calcul Excel. Dans la cellule A1, entrez `=UniformContDist(5 ; 3 ; 7 ; TRUE)` et vous obtiendrez 0.50. Ceci est bel et bien la fonction de répartition d'une loi uniforme centrée sur l'intervalle [3,7] évalué à 5. Entrez la même fonction, mais remplacez TRUE par FALSE et vous obtiendrez 0.25, soit la fonction de densité. Vous pouvez aussi insérer cette fonction dans une cellule en allant dans le menu Insertion, Fonction. Vous trouverez `UniformContDist` dans la section Fonctions personnalisées.

N'oubliez pas qu'à l'intérieur de cette fonction vous avez la liberté de tout créer comme si c'était une autre fonction ou une autre routine, rendant cette fonctionnalité de Visual Basic, très utile dans plusieurs situations.

Déclaration de variables de type *Object*

Jusqu'à maintenant, nous avons vu qu'une variable pouvait stocker des nombres, des chaînes de caractères, des dates, etc. Il est aussi possible de stocker dans une variable un objet et ses propriétés. Une fois l'objet stocké, cette variable se manipule comme si c'était un objet standard.

Par contre, l'assignation dans une variable objet ne se fait pas comme à l'habitude. Il faut utiliser la commande `Set`.

Déclaration de variables de type *Object*

Syntaxe

```
Dim VariableObjet as NomObjet  
Private VariableObjet as NomObjet
```

Exemple

La feuille avec laquelle nous travaillons le plus souvent est la feuille «Principale». Par contre, plusieurs données vitales sont stockées dans la feuille «Donnees» et il est souvent nécessaire d'y faire référence. Il est possible d'utiliser la structure `With... End With`, par contre il faut utiliser celle-ci à chaque référence. Nous utiliserons donc une variable de type `Objet` pour y faire référence. À suivre un peu plus loin pour stocker l'objet dans la variable.

```
Dim MaPlageRef as Range
```

Assignation dans une variable de type *Object*

Syntaxe

```
Set VariableObjet = ObjetDeTypeNomObjet
```

Exemple (suite)

```
Set MaPlage = ActiveWorkbook.Worksheets("Donnees").Range("A1:C1")
```

Remarque

`ObjetDeTypeNomObjet` doit être une instruction qui retourne un objet de type `NomObjet`.

Méthode *InputBox*

Nous avons vu un peu plus tôt que la fonction `InputBox` permettait à l'utilisateur de saisir une information lorsque celle-ci était demandée. Cette information était stockée directement dans une variable standard. La méthode `InputBox` quant à elle permet à l'utilisateur de saisir une information qui est considérée par Visual Basic comme étant un objet. Par exemple, cette information peut être une plage une cellule, une formule, etc.

Syntaxe

```
Application.InputBox(Prompt, Title, Default, Left, Top, HelpFile, HelpContextId,  
Type)
```

Arguments

<code>Prompt</code>	(Requis) Représente le texte qui sera affiché. Par exemple, «Sélectionnez une plage de cellules».
<code>Title</code>	(Optionnel) Représente le titre de la boîte de dialogue.
<code>Default</code>	(Optionnel) Représente la valeur par défaut dans la zone de saisie.
<code>Left</code>	(Optionnel) Position horizontale de la boîte de dialogue par rapport au coin supérieur gauche de l'écran.

Top	(Optionnel) Position verticale de la boîte de dialogue par rapport au coin supérieur gauche de l'écran.
HelpFile	(Optionnel) Spécifie le nom du fichier d'aide associé à la boîte de dialogue
HelpContextId	(Optionnel) Spécifie l'identification de la page d'aide associé au fichier d'aide spécifié avec HelpFile.
Type	(Optionnel) Type de données allant être entrées par l'utilisateur. Représente un nombre.

Type de données acceptées par la méthode *InputBox*

Valeur de type	Type de donnée retourné par <i>InputBox</i>
0	Formule
1	Valeur numérique
2	Chaîne de caractères
4	Valeur logique (Vrai ou Faux)
8	Objet de type Range
16	Valeur d'erreur
64	Tableau de valeurs (<i>Array</i>)

Exemple

Nous allons bâtir une boîte de dialogue qui permettra à l'utilisateur de spécifier une plage de cellules.

```
Sub Dialogue()
    Dim MaPlage As Range    'Déclare une variable de type objet Range

    Set MaPlage = Application.InputBox(Prompt:="Sélectionnez une plage de
cellules", Title:="Boîte de dialogue", Type:=8)    'Assigne dans la variable
objet MaPlage la plage saisie par l'utilisateur

    'On peut donc travailler avec la plage de cellules spécifiée par
l'utilisateur.
    MsgBox MaPlage.Cells(2, 1)    'Affiche l'élément situé à la 2e ligne, 1ère
colonne de la plage spécifiée par l'utilisateur
    MaPlage.Font.Size = 20    'Applique la taille de police 20 à la plage
spécifiée par l'utilisateur.
End Sub
```

Anciens travaux pratiques de l'année 2000

Cette section comprend les anciens travaux pratiques du cours Basic et Visual Basic avec la solution complète. Ils sont inclus dans le seul but de comprendre que le langage Visual Basic pour Excel est le même que le langage Visual Basic.

La méthode de documentation présentée dans cette section est celle suggérée par M. Marcel Dupras.

Travail pratique # 1

Contexte

On voudrait calculer l'espérance de la variable aléatoire Z qui se définit comme étant,

$$E[Z] = \frac{E[X]}{E[Y]}.$$

La variable aléatoire X peut seulement prendre les valeurs 1 et 2, tandis que la variable aléatoire Y peut prendre les valeurs 2 et 4. Vous devez demander à l'utilisateur, en utilisant la fonction *InputBox*, de fournir $\Pr(X=1)$ et $\Pr(Y=2)$. Votre programme devra calculer le reste. La réponse devra être affichée à l'aide d'un *MsgBox*.

Solution

Sub Esperance()

!*****

'Description du programme : Calculer l'espérance de la variable Z à partir de l'espérance de X et Y .
'L'espérance de X et Y sont calculées à partir de deux probabilités saisies par l'utilisateur.

'Pré-conditions : L'utilisateur devra fournir pour $P(X=1)$ et $P(Y=2)$ des valeurs de probabilités situées entre 0 et 1.
'Post-conditions : Le résultat final sera un nombre, qui correspondra à l'espérance de Z et sera affiché.

!*****

!*****

' Déclaration des variables

' Variables dont le contenu est fourni par l'utilisateur

Dim v_px1 As Double 'P(X=1) : variable d'entrée (Input) ==> probabilité que l'événement X prenne la valeur 1
Dim v_py2 As Double 'P(Y=2) : variable d'entrée (Input) ==> probabilité que l'événement Y prenne la valeur 2

' Variables dont le contenu est calculé

Dim v_px2 As Double 'P(X=2) : variable intermédiaire de traitement ==> probabilité que l'événement X prenne la valeur 2

Dim v_py4 As Double 'P(Y=4) : variable intermédiaire de traitement ==> probabilité que l'événement Y prenne la valeur 4

Dim v_ex As Double 'E(X) : variable intermédiaire de traitement ==> espérance mathématique de l'événement X

Dim v_ey As Double 'E(Y) : variable intermédiaire de traitement ==> espérance mathématique de l'événement Y

Dim v_ез As Double 'E(Z) : variable de sortie (Output) ==> espérance mathématique de l'événement Z

' Les valeurs possibles de X et Y auraient pu être entrées dans des constantes à l'aide de la commande Const

!*****

' Si on suppose que la pré-condition est satisfaite...

!*****

' 1ère étape : Obtenir les données de l'utilisateur

' Données fournies par l'utilisateur

' - P(X=1)

' - P(Y=2)

' Entrée des données par l'utilisateur

On Error GoTo Erreur 'Gestion des erreurs

1: v_px1 = InputBox("Veuillez entrer la probabilité que X soit égal à 1." & Chr(10) & "P(X=1) ?" & Chr(10) & "N.B. P(X=1) doit être compris entre 0 et 1.", "P(X=1) ?", "0,5")

Select Case v_px1 ' Validation de la probabilité de X=1

Case Is < 0

MsgBox "La valeur de P(X=1) doit être supérieure à 0.", vbOKOnly + vbExclamation, "Erreur !"

GoTo 1:

Case Is > 1

MsgBox "La valeur de P(X=1) doit être inférieure à 1.", vbOKOnly + vbExclamation, "Erreur !"

GoTo 1:

Case Else

GoTo 2:

End Select

!*****

2: v_py2 = InputBox("Veuillez entrer la probabilité que Y soit égal à 2." & Chr(10) & "P(Y=2) ?" & Chr(10) & "N.B. P(Y=2) doit être compris entre 0 et 1.", "P(Y=2) ?", "0,5")

Select Case v_py2 ' Validation de la probabilité de Y=2

Case Is < 0

MsgBox "La valeur de P(Y=2) doit être supérieure à 0.", vbOKOnly + vbExclamation, "Erreur !"

GoTo 2:

Case Is > 1

MsgBox "La valeur de P(Y=2) doit être inférieure à 1.", vbOKOnly + vbExclamation, "Erreur !"

GoTo 2:

Case Else

GoTo 3:

End Select

' Avec plus de notions en programmation, au lieu de supposer que les valeurs entrées sont correctes, le programme ferait

' lui-même cette vérification ==> Chose qui a été faite avec les Select Case

' P(X=1) est stocké dans v_px1 et P(Y=2) est stocké dans v_py2

!*****

!*****

' 2e étape : Calculs...

' a) P(X=2)

' b) P(Y=4)

' c) E(X)

' d) E(Y)

' e) E(Z)

' Données calculées et autogénérées

' - P(X=2) = 1 - P(X=1)

' - P(Y=4) = 1 - P(Y=2)

' - Espérance de X $E(X) = 1 \cdot P(X=1) + 2 \cdot P(X=2)$

' - Espérance de Y $E(Y) = 2 * P(Y=2) + 4 * P(Y=4)$

' - Espérance de Z $E(Z) = E(X)/E(Y)$

3: v_px2 = 1 - v_px1 'a)

v_py4 = 1 - v_py2 'b)

v_ex = v_px1 + 2 * v_px2 'c)

v_ey = 2 * v_py2 + 4 * v_py4 'd)

v_ez = v_ex / v_ey 'e)

' En calculant $P(X=2)$ et $P(Y=4)$ en faisant respectivement $1 - P(X=1)$ et $1 - P(Y=2)$, il est évident que la somme des

' probabilités pour X et Y va donner 1 pour chacun.

' $P(X=2)$ est stocké dans v_px2

' $P(Y=4)$ est stocké dans v_py4

' E(X) est stocké dans v_ex

' E(Y) est stocké dans v_ey

' E(Z) est stocké dans v_ez ==> sera le résultat affiché en 3.

' 3. Afficher le résultat

MsgBox "L'espérance mathématique de Z = " & v_ez, vbInformation + vbOKOnly, "E(Z) = " & Round(v_ez, 2)

Exit Sub

'Post-condition satisfaite : Le résultat (espérance de Z) a été affiché et correspond à un nombre.

' Gestion des erreurs

Erreur: MsgBox "Le programme a renvoyé une erreur." & Chr(10) & "L'utilisateur n'a pas entré un nombre ou a appuyé sur «Annuler»", vbOKOnly + vbExclamation, "Erreur inattendue !"

End Sub

Travail pratique # 2

Contexte

Votre programme devra calculer la valeur actuelle d'un seul paiement, en connaissant le taux d'intérêt et le temps. Ces données doivent être entrées par l'utilisateur dans une feuille de calcul Excel. Avant d'exécuter le calcul, le programme doit avertir (*MsgBox*) l'utilisateur de la démarche à suivre pour assurer le bon fonctionnement du programme. Voici d'ailleurs ce à quoi il pourrait ressembler.

	A	B	C	D	E	F	G
1							
2		Montant	Taux d'intérêt	Nombre d'années		Valeur actuelle	
3		VF	i	n		VA	
4		15000	10	10		5783,15	
5							
6		Calculer					
7							
8							
9							
10							

Vous n'êtes pas obligés d'utiliser un bouton tel que sur l'illustration.

Solution

```

*****
'DESCRIPTION DU PROGRAMME : Renvoyer la valeur actuelle d'un paiement lorsque le taux d'intérêt est i
' et la durée est de n

'PRÉ-CONDITIONS :
'a) On suppose que l'utilisateur a compris le message d'avertissement et qu'il entrera les données dans les
bonnes cellules
'b) Le paiement vf à actualiser doit être un nombre réel positif
'c) Le taux d'intérêt i doit être un nombre réel positif, où 10% correspond au nombre 10.
'd) La durée n doit être aussi un nombre réel positif, où n est un nombre d'années et la partie décimale, des
fractions d'une année

'POST-CONDITION : La valeur actuelle du paiement est affichée dans 2 cellules à droite de la dernière, et
correspond à un
nombre réel positif.
*****

*****

'Déclarations des variables et constantes
Private p_vf As Double ' variable d'entrée (Input) ==> correspond à la valeur future du paiement à actualiser
Private p_i As Double ' variable d'entrée (Input) ==> correspond au taux d'actualisation (taux d'intérêt)
Private p_n As Double ' variable d'entrée (Input) ==> correspond au nombre d'années
Private p_va As Double ' variable de sortie (Output) ==> correspond à la valeur actuelle du paiement
*****

Sub PRINCIPAL()
  MsgBox "Pour bien utiliser ce programme il faut... " & Chr(10) & "1) Entrer le paiement à actualiser dans la
cellule B4. " & Chr(10) _
  & "2) Entrer le taux d'intérêt dans la cellule C4 sous cette forme : 10 pour 10%, 5 pour 5%, etc..." & Chr(10) &
"3) Entrer le nombre d'années dans la cellule D4.", vbOKOnly + vbExclamation, "Avertissement !"

'A: Pré-conditions a) à d) satisfaites

  Call Obtenir
  Call Calcul
  Call Afficher
End Sub

*****

' 1ère étape : Obtenir les données de l'utilisateur
Sub Obtenir()
  Range("A1").Activate
  p_vf = ActiveCell.Offset(3, 1).Value 'Aurait pu être : Range("B4").Value
  p_i = ActiveCell.Offset(3, 2).Value / 100 'Aurait pu être : Range("C4").Value / 100
  p_n = ActiveCell.Offset(3, 3).Value 'Aurait pu être : Range("D4").Value

'A: La valeur du paiement est stockée dans p_vf, le taux d'intérêt est stocké dans p_i, et le nombre d'années est
stocké dans p_n
End Sub

*****

' 2e étape : Calculer la valeur actuelle... en référence avec la fonction
Sub Calcul()
  p_va = ValActuelle(p_vf, p_n, p_i)

'A: La valeur actuelle du paiement est stockée dans p_va

```

End Sub

' 3e étape : Afficher la valeur actuelle
Sub Afficher()

 ActiveCell.Offset(3, 5) = Round(p_va, 2)

End Sub

' A: Post-condition satisfaite : la valeur actuelle du paiement est affiché dans la bonne cellule

' PRÉ-CONDITIONS DE LA FONCTION

' Le contenu de l'argument fv doit être un nombre réel positif

' Le contenu de l'argument n doit être un nombre réel positif

' Le contenu de l'argument i doit être un nombre réel positif, où 10% = 0,1

' POST-CONDITIONS DE LA FONCTION

' ValActuelle renvoie la valeur actuelle du paiement (fv) placé pendant n années au taux d'intérêt i.

' Il s'agit d'un nombre réel positif.

Function ValActuelle(fv, n, i)

' Description des arguments

' fv : correspond à la valeur future du paiement

' n : correspond au nombre d'années pour actualiser le paiement

' i : correspond au taux d'intérêt

' A: Pré-condition satisfaite car la pré-condition de la procédure suppose les mêmes pré-conditions que pour la

' fonction

 ValActuelle = fv * (1 + i) ^ (-n)

' A: Post-condition satisfaite : la valeur actuelle est calculée.

End Function

Travail pratique # 3

Contexte

Vous travaillez dans une compagnie d'assurance automobile et vous êtes chargés de la tarification d'un produit d'assurance. On vous demande de bâtir un programme qui calculera les primes selon certaines caractéristiques de l'assuré tels que l'âge, le sexe, et la ville. L'assuré peut aussi choisir certaines mesures de réduction du risque telles qu'une franchise (déductible) et une limite de couverture. On suppose que la distribution d'un sinistre est exponentielle avec équations...

$$f_X(x) = \lambda e^{-\lambda x}$$

La prime pure est donc

$$PP = \int_d^{d+L} (x-d)f_X(x)dx + \int_{d+L}^{\infty} Lf_X(x)dx$$

Avec les simplifications, on a,

$$PP = \frac{e^{-\lambda d} - e^{-\lambda(d+L)}}{\lambda}$$

L'assuré choisit ses modalités de couverture (limite et franchise). Selon ses caractéristiques, la prime brute est calculée de la façon suivante.

- Le type de ville habité par l'assuré détermine le paramètre lambda. Lambda vaut 1/10000 pour une petite ville, 1/15000 pour une moyenne ville et 1/20000 pour une grande ville.
- Ensuite, on calcule la prime pure (PP) avec les équations précédentes.
- L'âge et le sexe détermineront le facteur d'ajustement à appliquer à la prime. Le facteur de départ est 1.
- Un assuré masculin a une surprime de 10% et un assuré féminin, un rabais de 15% par rapport à la prime de base.
- Un assuré âgé entre :
 - 16 et 25 ans : surprime de 20%
 - 26 et 40 ans : aucune surprime
 - 41 à 59 ans : rabais de 10%
 - 60 ans à 100 ans : surprime de 10%
- Les facteurs sont additifs, c'est-à-dire qu'un homme de 20 ans a une surprime de (20%+10%=30%).
- Les éléments comme la franchise (compris entre 0 et 1000\$), la limite de couverture (compris entre 5000\$ et 50000\$), l'âge, le sexe et la ville doivent être obtenus à l'aide de 5 *InputBox*.
- La réponse doit être affichée avec un *MsgBox*.

Solution

'DESCRIPTION DU PROGRAMME : Le programme doit calculer la valeur de la prime d'assurance-automobile en tenant compte

' de certains facteurs tels que la limite de couverture, la franchise, la ville, l'âge et le sexe de l'assuré.

'PRÉ-CONDITIONS :

'a) La limite de couverture doit être un nombre entier positif compris entre 5 000 et 50 000\$

'b) La franchise doit être un nombre entier positif compris entre 0 et 1 000\$

'c) L'utilisateur doit entrer 1 s'il habite dans une petite ville, 2 s'il s'agit d'une moyenne ville et 3 dans le cas
' d'une grande ville

'd) L'âge doit être un entier positif compris entre 16 et 100.

'e) L'utilisateur doit entrer «M» ou «m» s'il est un homme, ou «F» ou «f» si elle est une femme.

'NOTE: PUISQUE LE PROGRAMME PRINCIPAL EST DIVISÉ EN SOUS-PROGRAMMES, SELON LES ÉTAPES DE L'ALGORITHME, LES PRÉ ET POST

'CONDITIONS S'APPLIQUENT DONC POUR TOUTES LES PROCÉDURES.

'POST-CONDITION : La valeur de la prime est un nombre réel positif affiché dans un *MsgBox*.

```

'Déclarations des variables et constantes
Private v_limite As Long 'variable d'entrée (EN) ==> contient la limite de couverture de l'assuré
Private v_deductible As Long 'variable d'entrée (EN) ==> contient la franchise de l'assuré
Private v_ville As Byte 'variable d'entrée (EN) ==> contient le numéro de la type de ville qu'habite l'assuré
Private v_age As Byte 'variable d'entrée (EN) ==> contient l'âge de l'assuré
Private v_sexe As String * 1 'variable d'entrée (EN) ==> contient le sexe de l'assuré (M ou m ou F ou f)

Private v_lambda As Double ' variable intermédiaire de calcul ==> contient le facteur lambda qui dépend du type
de ville
' habitée par l'assuré
Private v_facteur As Double 'variable intermédiaire de calcul ==> contient un facteur d'augmentation ou de
diminution de
' la prime selon l'âge ou le sexe de l'assuré

Private v_prime As Double ' variable de sortie (SO) ==> contient la prime calculée et ajustée par le facteur

*****
Sub PRINCIPAL() ' Procédure qui appelle toutes les sous-routines (étapes de l'algorithme)
    Call Obtenir
    Call Calcul
    Call Afficher
End Sub
*****
Sub Obtenir() ' 1ère étape : Obtenir les données de l'utilisateur

MsgBox "Bienvenue au programme de calcul des primes d'assurance-automobile" & Chr(10) & "Veuillez
répondre aux questions qui vont suivre", vbInformation + vbOKOnly, "Bienvenue !"

v_limite = InputBox("Veuillez entrer une limite de couverture d'assurance." & Chr(10) & "N.B. Cette limite doit
être comprise entre 5 000$ et 50 000$", "Limite de couverture", 50000)
'A: Pré-condition a) satisfaite. La limite de couverture est un nombre entier positif compris entre 5 000 et 50 000$
et
' est stocké dans v_limite

v_deductible = InputBox("Veuillez entrer le déductible (franchise)." & Chr(10) & "N.B. Cette franchise doit être
comprise entre 0$ et 1000$", "Franchise", 500)
'A: Pré-condition b) satisfaite. La franchise (déductible) est un nombre entier positif compris entre 0 et 1 000$ et
' est stocké dans v_deductible

v_ville = InputBox("Veuillez entrer le type de ville dans lequel vous vivez." & Chr(10) & "Choix de réponses :." &
Chr(10) & _
"1) Petite ville" & Chr(10) & "2) Moyenne ville" & Chr(10) & "3) Grande ville", "Type de ville", 1)
'A: Pré-condition c) satisfaite. Le type de ville est un nombre entier qui est soit 1, 2 ou 3 (selon la ville) et
' est stocké dans v_ville

v_age = InputBox("Veuillez entrer votre âge." & Chr(10) & "N.B. Vous devez être âgé entre 16 et 100 ans pour
être couvert", "Âge", 20)
'A: Pré-condition d) satisfaite. L'âge est un nombre entier positif compris entre 16 et 100 et est stocké dans
v_age

v_sexe = InputBox("Veuillez entrer votre sexe." & Chr(10) & "M pour masculin et F pour féminin", "Sexe", "m")
'A: Pré-condition e) satisfaite. Le sexe est une chaîne de caractère qui est soit «M», «m», «F» ou «f»
' et est stocké dans v_sexe
End Sub

*****
Sub Calcul() '2e étape : Calculer la prime selon les facteurs modifiant celle-ci

```

```
! *****
```

```
Select Case v_ville 'Détermination de la valeur de lambda selon le type de ville habitée par l'assuré
```

```
Case Is = 1
```

```
    v_lambda = 1 / 10000
```

```
Case Is = 2
```

```
    v_lambda = 1 / 15000
```

```
Case Is = 3
```

```
    v_lambda = 1 / 20000
```

```
End Select
```

```
' A: La valeur de lambda a été déterminée par le programme et est stockée dans la variable v_lambda
```

```
! *****
```

```
! *****
```

```
v_facteur = 1 'Le facteur de modification de la prime par défaut est de 1 (bref, aucun changement)
```

```
Select Case v_sexe ' Modification de la prime selon le sexe de l'assuré
```

```
Case Is = "m"
```

```
    v_facteur = v_facteur + 0.1
```

```
Case Is = "f"
```

```
    v_facteur = v_facteur - 0.15
```

```
Case Is = "M"
```

```
    v_facteur = v_facteur + 0.1
```

```
Case Is = "F"
```

```
    v_facteur = v_facteur - 0.15
```

```
End Select
```

```
Select Case v_age ' Modification de la prime selon l'âge de l'assuré
```

```
Case 16 To 25
```

```
    v_facteur = v_facteur + 0.2
```

```
Case 26 To 40
```

```
    v_facteur = v_facteur + 0
```

```
Case 41 To 59
```

```
    v_facteur = v_facteur - 0.1
```

```
Case 60 To 100
```

```
    v_facteur = v_facteur + 0.1
```

```
End Select
```

```
' A: La valeur de notre facteur de modification de la prime a été calculé par le programme et est stocké dans la
```

```
' variable v_facteur
```

```
! *****
```

```
' Calcul de la prime finale à l'aide de la fonction et du facteur
```

```
v_prime = Esperance(v_deductible, v_limite, v_lambda) * v_facteur
```

```
' A: Prime finale calculée et stockée dans la variable v_prime
```

```
End Sub
```

```
!*****
```

```
Sub Afficher() ' 3e étape : Afficher la prime à l'aide d'un MsgBox
```

```
    MsgBox "La prime d'assurance-automobile est de " & Round(v_prime, 2) & " $", vbOKOnly, "Prime"
```

```
' A: Post-condition satisfaite : la valeur de la prime est un nombre réel positif et a été affichée dans un MsgBox
```

```
End Sub
```

```
!*****
```

```
' PRÉ-CONDITIONS DE LA FONCTION
```

```
' Le contenu de l'argument «deductible» doit être un nombre entier compris entre 0 et 1000 et représente la franchise
```

```
' dans l'intégrale (calcul de la prime de base)
' Le contenu de l'argument «limite» doit être un nombre entier compris entre 5 000 et 50 000$ et représente la
limite
' de couverture dans l'intégrale (calcul de la prime de base)
' Le contenu de l'argument «lambda» peut prendre seulement trois valeurs et représente un certain facteur selon
la ville
' de l'assuré qui ajuste le risque de sinistre et ce, dans l'intégrale (calcul de la prime de base)
```

```
' POST-CONDITIONS DE LA FONCTION
```

```
' «Esperance» calcule la prime de base et représente un nombre réel positif
```

```
Function Esperance(deductible, limite, lambda)
```

```
' A: Pré-conditions satisfaites
```

```
    Esperance = (Exp(-deductible * lambda) - Exp(-lambda * (deductible + limite))) / (10 * lambda)
```

```
' A: Post-condition satisfaite : la prime de base a été calculée
```

```
End Function
```

```
' Résultat de l'intégrale (voir plus de détails dans les feuilles incluses avec le TP)
```

```
'  $E(x) = (e^{-deductible * lambda} - e^{-lambda * (deductible + limite)}) / lambda$ 
```

Travail pratique # 4

Contexte

Vous devez bâtir un programme qui va trouver, à l'aide de la méthode de la bissectrice, la valeur de la racine carrée de 2.

Cette méthode s'applique pour trouver le zéro d'une fonction. Trouver la racine carrée de 2 est équivalent à trouver le zéro de la fonction $f(x) = x^2 - 2 = 0$. La méthode de la bissectrice consiste à séparer plusieurs intervalles en 2. L'intervalle de départ doit être tel que la fonction peut être positive et négative dans cet intervalle. On sépare l'intervalle en 2 et le sous-intervalle dans lequel il y a changement de signe implique que le zéro de la fonction est dans cette zone. On répète cette étape plusieurs fois jusqu'à ce qu'on ait atteint une borne d'erreur maximale ou un maximum d'itérations.

L'utilisateur doit entrer les paramètres à l'aide d'*InputBox*, c'est-à-dire le nombre maximal d'itérations et l'erreur maximale permise.

Solution

```
Sub ZeroFonction()
```

```
' //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
' OBJECTIF : À l'aide de la méthode de la bisection, trouver le zéro de la fonction
```

```
'  $f(x) = x^2 - 2$ 
```

```
'
```

```
' PRÉ-CONDITIONS :
```

```
' a) Le nombre d'itérations maximal que l'utilisateur doit entrer doit être un nombre compris
dans [1..32767] (ENT_MAX)
```

```
' b) La marge d'erreur que l'utilisateur doit entrer doit être un nombre réel compris dans
[0..1]
```

```
'
```

```
' POST-CONDITION :
```

```
' Le programme renvoie un intervalle très petit dans lequel le «vrai» zéro de la fonction
est situé. À la limite, chaque borne de cet intervalle tend vers le zéro de la fonction.
```

```
' On pourrait définir «z» comme étant la valeur au milieu de cet intervalle qui sera un nombre
' réel.
' //////////////////////////////////////

' //////////////////////////////////////

' Déclaration des variables et constantes

Dim m_erreur As Double ' Variable d'entrée (EN) ==> correspond à la marge d'erreur maximale
' acceptée par l'utilisateur
Dim it_max As Integer ' Variable d'entrée (EN) ==> correspond au nombre de pas (itérations) que
' le programme doit exécuter
Dim n As Integer ' Variable intermédiaire de calcul ==> correspond au nombre d'itérations en cours
' Il s'agit d'un accumulateur
Dim a As Double ' Variable intermédiaire de calcul ==> correspond à la borne inférieure du grand
' intervalle
Dim b As Double ' Variable intermédiaire de calcul ==> correspond à la borne supérieure du grand
' intervalle
Dim x1 As Double ' Variable intermédiaire de calcul ==> correspond à la borne inférieure et
' supérieure des sous-intervalles
Dim binf As Double ' Variable intermédiaire de calcul ==> correspond à la borne inférieure de
' l'intervalle dans lequel est situé le zéro de la fonction
Dim bsup As Double ' Variable intermédiaire de calcul ==> correspond à la borne supérieure de
' l'intervalle dans lequel est situé le zéro de la fonction
Dim z As Double ' Variable de sortie (SO) ==> correspond au zéro de la fonction approximé
' A: Rien d'obtenu de l'utilisateur, pré-conditions a) et b) satisfaites
' //////////////////////////////////////

' **** 1ère étape : Obtenir les données de l'utilisateur ****

it_max = InputBox("Veuillez entrer le nombre maximal d'itérations" & Chr(10) & "Cette valeur doit être comprise
entre 1 et 32767", "Nombre maximal de pas", 32767)
m_erreur = InputBox("Veuillez définir la marge d'erreur" & Chr(10) & "La marge d'erreur doit être comprise entre
0 et 1", "Marge d'erreur maximale", 0.0000000001)
' A: 1 <= it_max <= 32767 où it_max est un entier qui contient le nombre de pas maximal,
' obtenue de l'utilisateur, stockée dans it_max
' A: 0 <= m_erreur <= 1 où m_erreur est un nombre réel qui contient la marge d'erreur maximale,
' obtenue de l'utilisateur, stockée dans m_erreur
' A: Aucun affichage du zéro de la fonction

' **** 2e étape : Calculer le zéro de la fonction à l'aide de la boucle ****

'A: Situation initiale (SI) ==> 4e étape : Conditions d'initialisation
a = 1 ' La valeur de la borne inférieure de l'intervalle initial est de 1
b = 2 ' La valeur de la borne supérieure de l'intervalle initial est de 2
x1 = (a + b) / 2 ' x1 est le point milieu de l'intervalle initial, correspondra aux bornes
' inférieures et supérieures des 2 sous-intervalles
n = 0 ' Le nombre d'itérations qui a été exécuté est de 0

' 1ère étape : Définir une hypothèse de récurrence
' A: HR
' a) a est toujours la borne inférieure du grand intervalle
' b) b est toujours la borne supérieure du grand intervalle
' c) x1 est toujours la borne inférieure et supérieure des sous-intervalles
' d) binf est la borne inférieure du sous-intervalle choisi (celui dans lequel est le zéro de la fonction)
' e) bsup est la borne supérieure du sous-intervalle choisi (celui dans lequel est le zéro de la fonction)
' f) it_max contient toujours le nombre maximal d'itérations
```

' g) m_erreur contient toujours la marge d'erreur maximale acceptée par l'utilisateur

' 2e étape : Définir la condition d'arrêt

' A: Conditions d'arrêt ==> la boucle s'arrêtera dès qu'une OU l'autre de ces conditions est atteinte

' - La valeur absolue de (a-b) est inférieure à la marge d'erreur entrée par l'utilisateur

' - Le nombre d'itérations exécutées est égal au nombre d'itérations maximal entré par l'utilisateur

' 3e étape : Concevoir le corps de l'itération

While (Abs(a - b) >= m_erreur) And (n < it_max)

' Début du progrès

' A: La valeur absolue de (a-b) est >= à la marge d'erreur

' et le nombre d'itérations exécutée est inférieur au nombre maximal de pas entré par l'utilisateur

' A: pré-conditions a), b), f) et g) de HR satisfaites

 If f(a) < 0 And f(x1) > 0 Then ' Vérification du sous-intervalle qui contient le zéro de la fonction

 binf = a

 bsup = x1

' A: Si le zéro de la fonction est compris dans cet intervalle, alors les bornes choisies sont a et x1

 Else

 binf = x1

 bsup = b

' A: Si le zéro de la fonction est compris dans cet intervalle, alors les bornes choisies sont x1 et b

 End If

' A: pré-conditions c) à e) de HR satisfaites

 a = binf

 b = bsup

 x1 = (a + b) / 2

 n = n + 1

' A: Les valeurs de a, b et x1 sont mises à jour suite à la sélection du bon sous-intervalle, afin que ce sous-intervalle

' soit redivisé en 2 autres sous-sous-intervalles

 ' Fin du progrès

Wend

' A: La valeur absolue de (a-b) est < à la marge d'erreur

' OU le nombre d'itérations exécutée est égal au nombre maximal de pas entré par l'utilisateur

z = x1

' A: Le zéro de la fonction est stocké dans la variable z

' A: Situation finale (SF)

' m_erreur et it_max n'auront pas changé durant l'itération

' a correspondra à la valeur de la borne inférieure de l'intervalle choisi ==> tendra vers le zéro de la fonction

' b correspondra à la valeur de la borne supérieure de l'intervalle choisi ==> tendra vers le zéro de la fonction

' x1 correspondra à la valeur de la borne supérieure de l'intervalle choisi ==> sera la moyenne des 2 bornes

' n correspondra au nombre d'itérations nécessaires pour renvoyer un zéro de la fonction qui respecte

' les conditions de la boucle

' z contiendra le zéro de la fonction

' **** 3e étape : Afficher le zéro de la fonction ****

MsgBox "Le zéro de la fonction $f(x) = x^2 - 2$ est de : " & Chr(10) & Round(z, 9), vbInformation, "Zéro de la fonction en " & n & " itérations"

' A: Post-condition satisfaite : le zéro de la fonction a été affiché et correspond à un nombre réel.

```

End Sub

' ///////////////////////////////////////////////////////////////////
' OBJECTIF DE LA FONCTION
' Calculer la valeur de  $f(x) = x^2 - 2$ 
'
' PRÉ-CONDITIONS DE LA FONCTION
' Le contenu de l'argument «x» doit être n'importe quel nombre réel
'
' POST-CONDITIONS DE LA FONCTION
' «f» calcule la valeur  $f(x) = x^2 - 2$  lorsque x est un nombre réel, et renvoie f(x)
Function f(x)
    ' A: Pré-conditions satisfaites
    f = x ^ 2 - 2
    ' A: Post-condition satisfaite : la valeur de f(x) a été calculée
End Function

```

Travail pratique # 5

Contexte

Vous devez bâtir un programme qui calcule et retourne le nombre de façons que l'on peut distribuer " n " balles distinctes dans " r " urnes. Si les capacités de l'ordinateur sont dépassées, aucune erreur VB doit apparaître ; seul un message provenant du programme doit avertir l'utilisateur.

Solution

```
Sub Répartition()
```

```

' Description :
' Le programme calcule et retourne le nombre de façons que l'on peut distribuer " n " balles distinctes dans " r " urnes.

' Pré-condition :
' 1. L'utilisateur fournit un entier " n " dans l'intervalle [1..ENTIER_MAX] représentant le nombre de balles à répartir.
' 2. L'utilisateur fournit un entier " r " dans l'intervalle [1..ENTIER_MAX - n + 1] représentant le nombre d'urnes.
' 3. " n " doit être plus grand ou égale à " r ".

' Post condition :
' 1. " n " et " r " sont les entiers fournis par l'utilisateur.
' 2. Si les calculs ne dépassent pas la capacité de l'ordinateur, le programme a affiché le nombre de façons qu'on peut distribuer les " n " balles dans les " r " urnes.
' 3. Si les calculs dépassent la capacité de l'ordinateur, le programme a affiché " Entier trop grand ".

' Déclaration des variables et constantes
Dim nballes As Long ' Variable d'entrée (EN) ==> contient le nombre de balles (compris entre 1 et ENTIER_MAX)
Dim urnes As Long ' Variable d'entrée (EN) ==> contient le nombre d'urnes (compris entre 1 et ENTIER_MAX - balles + 1)
Dim nfacons As Long ' Variable de sortie (SO) ==> contient le nombre de façons d'organiser n balles dans r urnes.. est un entier compris dans 1.. ENTIER_MAX
' A: Rien d'obtenu de l'utilisateur, pré-conditions 1 à 3 satisfaites

' **** 1ère étape : Obtenir les données de l'utilisateur ****

```

```

nballes = InputBox("Veuillez entrer le nombre de balles, compris entre 1 et ENTIER_MAX", "Nombre de balles")
nurnes = InputBox("Veuillez entrer le nombre d'urnes, compris entre 1 et ENTIER_MAX - Balles + 1", "Nombre
d'urnes")
' A: 1 <= nballes <= ENTIER_MAX où nballes est un entier qui contient le nombre de balles, a été obtenue de
l'utilisateur
' A: 1 <= nurnes <= (ENTIER_MAX - nballes + 1) où nurnes est un entier qui contient le nombre d'urnes, a été
obtenue de l'utilisateur
' A: Aucun affichage du nombre de façons

' **** 2e étape : Calculer le nombre de façons ****
' Vérification si la fonction Factorielle a renvoyé 0 (Entier trop grand)...
If (Factorielle(nballes) = 0) Or (Factorielle(nurnes - 1) = 0) Or (Factorielle(nurnes + nballes - 1) = 0) Then
' **** 3e étape : Afficher le résultat ****
    MsgBox "Entier trop grand", vbExclamation + vbOKOnly, "Erreur !"
' A: Post-condition satisfaite : la fonction a dépassé la capacité, le programme a renvoyé "Entier trop grand"
Else
' Si Factorielle n'a pas renvoyé 0 pour chaque terme de la combinaison, nfacons sera bien calculé
nfacons = Factorielle(nballes + nurnes - 1) / (Factorielle(nballes) * Factorielle(nurnes - 1))
' **** 3e étape : Afficher le résultat ****
    MsgBox "Il y a " & nfacons & " façons de répartir " & nballes & " balles dans " & nurnes & " urnes",
vbInformation + vbOKOnly, "Résultat"
' A: Post-condition satisfaite : la fonction a pu calculer les factorielles, le nombre de facons a été affiché
End If
End Sub
Function Factorielle(k As Long) As Long
' Pré-condition :
' Le paramètre " k " reçu de la procédure appelante doit être un entier dans [0..ENTIER_MAX].

' Post-condition :
'1. L'entier " k " est celui reçu en paramètre.
'2. Si k = 0, la fonction retourne 1.
'3. Si k > 0, la fonction retourne k! si les calculs ne dépassent pas la capacité de l'ordinateur et elle retourne 0
s'ils la dépassent.
'4. L'exécution de cette fonction ne modifie rien d'autre à l'extérieur de celle-ci.

' Déclaration des variables et constantes
Const ENTIER_MAX As Long = 2147483647 ' Contient la valeur maximale stockée dans un LONG
Dim Test As Long ' Contient une valeur inférieure à ENTIER_MAX... permet de valider si k! va entrer dans
ENTIER_MAX
Dim CpteurTest As Long ' Contient le compteur ou diviseur pour valider mon test
Dim n As Long ' Contient le compteur pour calculer la factorielle... n * (n-1) *...

' A: Pré-condition satisfaite

Select Case k
Case Is = 0
    Factorielle = 1
' A: Post-condition 2 satisfaite
Case Is = 1
    Factorielle = 1
Case Else
' Tests si l'on peut calculer la factorielle de k
' Principe : Si on divise ENTIER_MAX par k! et que le résultat appelé Test est >= à 1, alors k! entre dans
ENTIER_MAX
' sinon, elle va défoncer !

' 4e étape : Condition d'initialisation
' A: Conditions d'initialisation de ma boucle de test

```

```

CpteurTest = k ' Compteur ou diviseur au départ est égal à k
Test = ENTIER_MAX / CpteurTest ' Test au départ est égal à ENTIER_MAX / CpteurTest

' 1ère étape : Définir l'hypothèse de récurrence
' A: HR
' - Test est toujours la valeur de référence pour savoir si k! va dépasser les capacités de l'ordinateur
' - CpteurTest est toujours la valeur qui va diviser Test
' Ces deux variables sont toujours mises à jour tout au long de l'itération

' 2e étape : Définir la condition d'arrêt
' A: Condition d'arrêt : CpteurTest doit être inférieur ou égal à 1 car je divise par 3, par 2... et j'arrête.
Pourquoi diviser par 1
' ET la boucle s'arrête dès que Test est inférieure à 1 car on sait que si Test < 1, alors on dépasse les
capacités de l'ordinateur

' 3e étape : Corps de l'itération
While (CpteurTest > 1) And (Test >= 1)
' Début du progrès
' A: CpteurTest est > à 1 et Test est >= à 1
CpteurTest = CpteurTest - 1 ' Compteur qui diminue par CpteurTest, par CpteurTest-1, par CpteurTest-2,
etc...
Test = Int(Test / CpteurTest) ' Valeur entière car toutes les variables sont de type LONG
' A: Pré-conditions de HR satisfaites
' Fin du progrès
Wend
' A: CpteurTest <= à 1 OU Test < 1

If Test >= 1 Then ' Le test est nécessairement concluant
' A: Initialisation de l'itération... Factorielle = 1
Factorielle = 1
For n = 2 To k
Factorielle = Factorielle * n ' Incrémentement de n à chaque itération.. sur le principe de la factorielle
Next n ' Incrémentement du compteur automatique avec n
' A: Puisque le test a été concluant, Factorielle contient k!
Else
Factorielle = 0
' A: La fonction renvoie 0 si elle a dépassé la capacité.. si Test < 1, Factorielle = 0
End If
End Select

' A: Post-condition 4 satisfaite
End Function

```

Références

Bidault, Mikaël. Excel 2000 & Visual Basic pour Applications 6. Campus Press, Simon & Schuster Macmillan. 1999. 534 pages.

Aide en ligne de Microsoft Excel et Visual Basic pour Excel.

Notes de cours, Basic et Visual Basic, Année 2000.