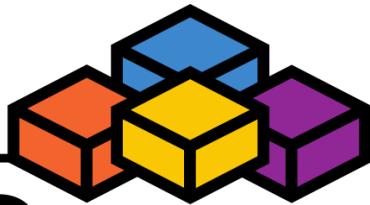




INSTITUT UNIVERSITAIRE D'ABIDJAN

Microsoft **Visual Basic** for Applications



Programmation VBA

Avec Excel

M1 GESTION DES RISQUES EN ASSURANCE ET EN FINANCE

Emmanuel Evilafou

Année 2024-2025

Version du 10 Février 2025

Table des matières

Introduction à VBA	6
L'onglet Développeur.....	6
L'enregistreur de macros	7
Définition d'une première macro	8
Tester la macro	10
Modifier une macro	11
Affecter une macro à une icône dans le ruban	15
Affecter une macro à un bouton de formulaire	16
Affecter une macro à un bouton de commande.....	19
Premiers pas en VBA	24
Si vous n'avez aucune idée de ce qu'est un langage objet.....	26
Propriétés	27
Méthodes	27
Procédures	27
Une première procédure	28
Fonctions	31
Commentaires.....	33
Commentaires	33
Variables.....	34
Variables Static.....	36
Cellules, plages, feuilles et classeurs	38
Tableaux	38
Cellules, plages, feuilles et classeurs.....	40
Le modèle objet d'Excel.....	41
Les principaux objets Excel.....	41
Les objets d'Excel – Quelques exemples pour bien comprendre.....	41
Les propriétés et les méthodes des objets d'Excel	42
Travailler avec des sélections	47
Travailler avec des couleurs	49
Utilisation des couleurs prédéfinies.....	49
Tests.....	50
If Then Else	50
Opérateurs logiques	54
Test multiple Select Case	54
Boucles	56

La boucle For ... Next.....	56
La boucle While ... Wend.....	61
La boucle Do While ... Loop	62
La boucle Do ... Loop While	62
La boucle Do Until ... Loop.....	63
Quitter une boucle prématurément	63
Gestion d'erreurs en VBA	63
Mise en place d'un gestionnaire d'erreurs.....	64
Annulation du gestionnaire d'erreurs	64
Reprise de l'exécution	65
Poursuite de l'exécution	66
Messages.....	67
La méthode MsgBox().....	67
La fonction MsgBox()	70
La fonction InputBox().....	72
Traitement du texte.....	73
Longueur d'une chaîne	73
Extraction de sous-chaînes	73
Suppression d'espaces au début et/ou à la fin	74
Casse des caractères	75
Valeur numérique d'une chaîne.....	75
Remplacement d'une sous-chaîne par une autre	76
Conversions et mises en forme.....	77
Mise en forme.....	77
Longueur d'une chaîne	78
Extraction de sous-chaînes	79
Suppression d'espaces au début et/ou à la fin	79
Casse des caractères	80
Valeur numérique d'une chaîne.....	80
Remplacement d'une sous-chaîne par une autre	81
Conversions et mises en forme.....	82
Mise en forme.....	82
Traitement des nombres	83
Opérateurs mathématiques	83
Opérateurs de comparaison	84
Opérateurs logiques	85

Fonctions mathématiques	85
Fonctions de conversion	86
Mise en forme	86
Traitement des dates	87
Fonctions de base dédiées aux dates	87
Les fonctions IsDate() et CDate().....	88
Les fonctions de date qui travaillent sur des intervalles.....	89
Mise en forme.....	93
Traitement des classeurs.....	94
Ouvrir un classeur vierge.....	94
Ouvrir un classeur.....	94
Fermer un classeur.....	95
Exécuter une procédure à la fermeture d'un classeur.....	99
Modifier la mise en forme des cellules.....	100
Couleur du texte	100
Police et attributs	102
Format des cellules	103
Copier-coller en VBA.....	106
Utiliser les fonctions d'Excel en VBA	107
Min, Max et Average.....	107
Nombre de cellules supérieures à 20.....	108
UserForms	108
Définition d'une UserForm.....	108
Personnalisation de la boîte de dialogue personnalisée.....	109
Insertion de contrôles dans une boîte de dialogue personnalisée.....	112
Labels dans une boîte de dialogue personnalisée	112
TextBox dans une boîte de dialogue personnalisée	114
Des ComboBox dans une boîte de dialogue personnalisée.....	116
Des ListBox dans une boîte de dialogue personnalisée	117
Des CheckBox dans une boîte de dialogue personnalisée	120
Des OptionButton dans une boîte de dialogue personnalisée	121
Le contrôle MultiPage dans une boîte de dialogue personnalisée.....	123
Des SpinButton dans une boîte de dialogue personnalisée	124
Des CommandButton dans une boîte de dialogue personnalisée	126
Des calculs dans un UserForm	127
Des images dans une boîte de dialogue personnalisée.....	128

Tracé d'un graphique en VBA	132
Agir sur les propriétés d'un classeur	134
Afficher un message dans la barre d'état	137
Dim et Set	138
Définir un nouveau classeur, une nouvelle feuille	139
Ajouter un graphique personnalisé.....	140
Changer la couleur de certaines cellules en fonction de leur valeur	145
Exécuter une procédure à une certaine heure	146
Exécution d'une procédure à l'appui sur une touche ou une combinaison de touches	148
Copier, renommer et supprimer un fichier	151
Lister les fichiers contenus dans un dossier	152
Nombre de fichiers contenus dans un dossier	153
Tester si un fichier existe.....	154
Sauvegarde dans un fichier texte.....	155
Un premier exemple	156
Copie de la plage A1:A5 dans un fichier texte	157
Lecture d'un fichier texte	158
Manipuler des dossiers	160
Créer un dossier.....	161
Copier un dossier dans un autre.....	165
Déplacer un dossier dans un autre	165
Modifier et lire les attributs des fichiers.....	166
Modifier les attributs d'un fichier.....	166
Accès aux dossiers spéciaux.....	167
Recherche de doublons dans une colonne	169
Exercices.....	171

Introduction à VBA

Qu'est-ce que VBA ?

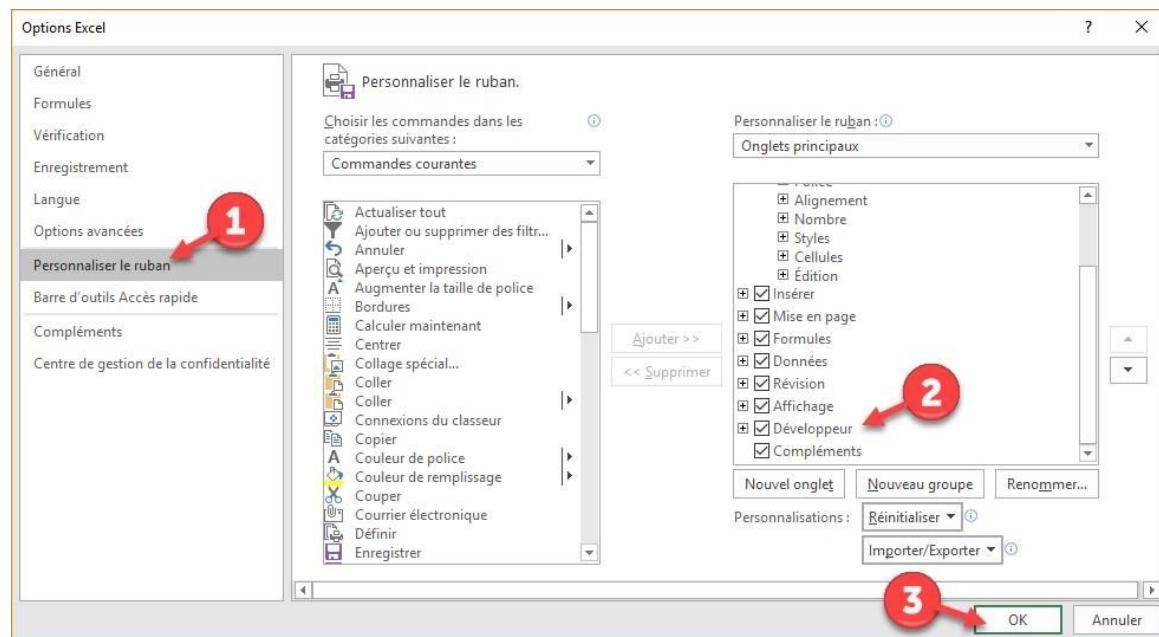
Il arrive que Excel ne dispose pas de fonctionnalités prédéfinies pour répondre à un besoin spécifique. Dans ce cas, il est nécessaire de « programmer Excel » afin de créer la fonctionnalité requise. Pour ce faire, vous pouvez utiliser soit l'enregistreur de macros, soit le langage VBA.

VBA est un langage de programmation intégré dans les applications Microsoft, telles qu'Excel, pour automatiser des tâches répétitives et développer des applications personnalisées. En finance, le VBA est utilisé pour effectuer des calculs complexes, générer des rapports financiers, analyser des données de marché, etc. Ce cours couvre les bases et des applications avancées de VBA pour la finance.

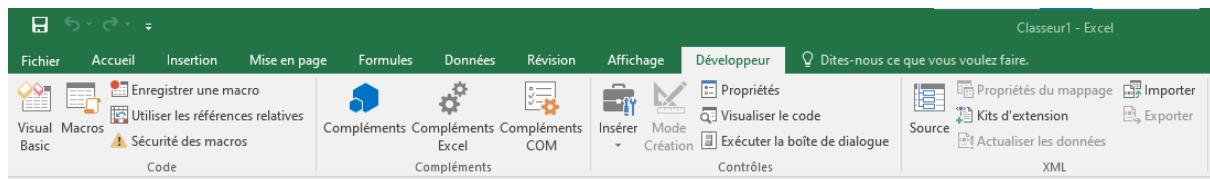
L'onglet Développeur

L'onglet **Développeur** du ruban d'Excel contient plusieurs icônes nécessaires lorsque l'on développe du code VBA. Cet onglet n'étant pas accessible par défaut dans le ruban, voyons comment le faire apparaître.

Lancez la commande **Options** dans le menu **Fichier**. La boîte de dialogue **Options Excel** s'affiche. Basculez sur l'onglet **Personnaliser le ruban** (1), cochez la case **Développeur** (2) et validez en cliquant sur **OK** (3).



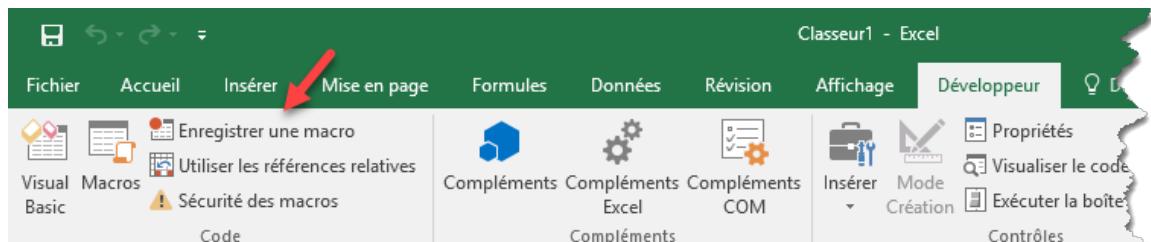
L'onglet **Développeur** fait désormais partie du ruban d'Excel. Vous l'utiliserez pour tous vos développements VBA :



L'enregistreur de macros

Avant d'aborder le langage VBA, je vous propose de découvrir l'enregistreur de macros. Cet outil enregistre les actions effectuées dans Excel et les transforme en des instructions VBA. Une fois l'enregistreur lancé, tout ce que vous faites au clavier et à la souris est enregistré. Par exemple, la frappe des touches, le clic sur une cellule, le clic sur des icônes du ruban, la mise en forme des cellules, lignes et colonnes, etc.. Lorsque vous arrêtez l'enregistreur, des instructions VBA sont générées et enregistrées dans le classeur. Par la suite, vous pourrez exécuter la macro autant de fois que vous le souhaitez en utilisant une icône dans l'onglet **Développeur** du ruban ou un raccourci-clavier.

Pour enregistrer une nouvelle macro, sélectionnez l'onglet **Développeur** dans le ruban, puis cliquez sur l'icône **Enregistrer une macro** dans le groupe **Code** :



Cette action déclenche l'ouverture de la boîte de dialogue **Enregistrer une macro** :



Dans cette boîte de dialogue :

- Définissez le nom de la macro, sans espace. Si le nom de la macro est composé de plusieurs mots, vous pouvez commencer chaque mot par une majuscule ou séparer les mots entre eux par des caractères de soulignement (_).
- Affectez si nécessaire un raccourci clavier à la macro pour faciliter son exécution.
- La liste déroulante **Enregistrer la macro dans** est initialisée par défaut à **Ce classeur**.

La macro sera donc liée au classeur courant et ne pourra s'exécuter que dans ce classeur. Vous pouvez également choisir de l'enregistrer dans le classeur de macros personnelles. Dans ce cas, la macro est enregistrée dans le classeur masqué **xlbs**, qui se trouve dans le dossier

C:\Utilisateurs\nom d'utilisateur\AppData\Local\Microsoft\Excel\XLStart. Elle sera disponible dans tous les classeurs. Les classeurs stockés dans le dossier **XLStart** s'ouvrent automatiquement à chaque démarrage d'Excel, et les éventuelles macros enregistrées dans le classeur de macros personnelles sont automatiquement accessibles.

- Décrivez la macro en quelques lignes dans la zone **Description**.

Cliquez enfin sur **OK** lorsque vous êtes prêt à lancer l'enregistrement.

Une première macro Excel

Définition d'une première macro

A titre d'exemple, nous allons définir une macro qui met en forme des dates :



11-janv	12-janv	13-janv	14-janv	15-janv
Samedi 11 janvier 2025	Dimanche 12 janvier 2025	Lundi 13 janvier 2025	Mardi 14 janvier 2025	Mercredi 15 janvier 2025

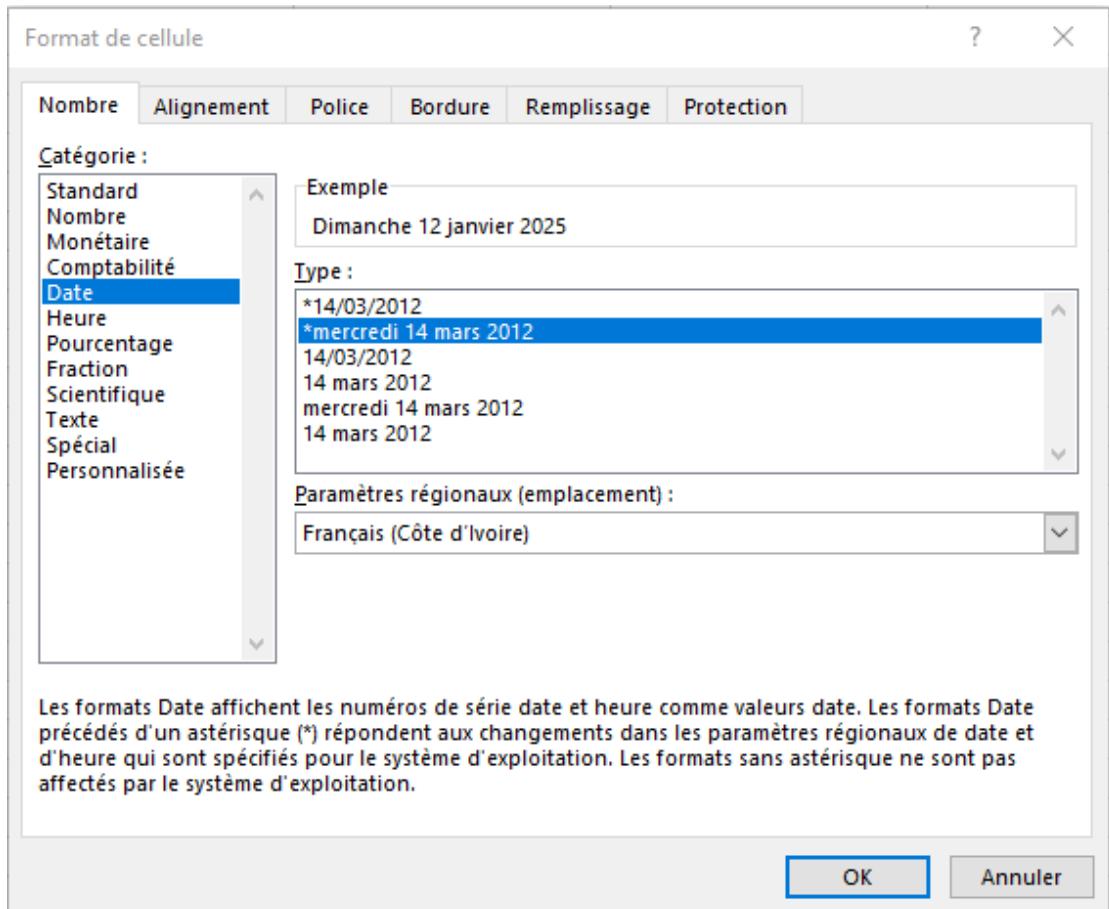
Avant de sélectionner l'onglet **Développeur** et de cliquer sur l'icône **Enregistrer une macro** dans le groupe **Code**, sélectionnez les cellules dont le format doit être changé :

	A	B	C	D	E	F
25						
26						
27		11-janv	12-janv	13-janv	14-janv	15-janv
28						
29						
30						

Puis cliquez sur l'icône **Enregistrer une macro**. La macro aura pour nom **DateLongue**. Elle sera accessible avec le raccourci clavier *Contrôle + Majuscule + D*, et elle sera stockée dans le classeur courant :



Un clic sur le bouton **OK** et l'enregistrement commence. Pour arriver au résultat recherché, le plus simple consiste à basculer sur l'onglet **Accueil** et à cliquer sur le lanceur de boîte de dialogue **Nombre**. La boîte de dialogue **Format de cellule** s'affiche. Sélectionnez **Date** dans la zone de liste **Catégorie** et le type **Date Longue** dans la zone de liste **Type**, puis cliquez sur **OK** :



Les formats Date affichent les numéros de série date et heure comme valeurs date. Les formats Date précédés d'un astérisque (*) répondent aux changements dans les paramètres régionaux de date et d'heure qui sont spécifiés pour le système d'exploitation. Les formats sans astérisque ne sont pas affectés par le système d'exploitation.

Vous allez maintenant stopper l'enregistrement de la macro. Basculez sur l'onglet **Développeur** dans le ruban puis cliquez sur **Arrêter l'enregistrement** dans le groupe **Code**.

Tester la macro

Pour vérifier que la macro fonctionne, commencez par saisir quelques dates dans la feuille de calcul où a été définie la macro ou dans une autre feuille de calcul du classeur :

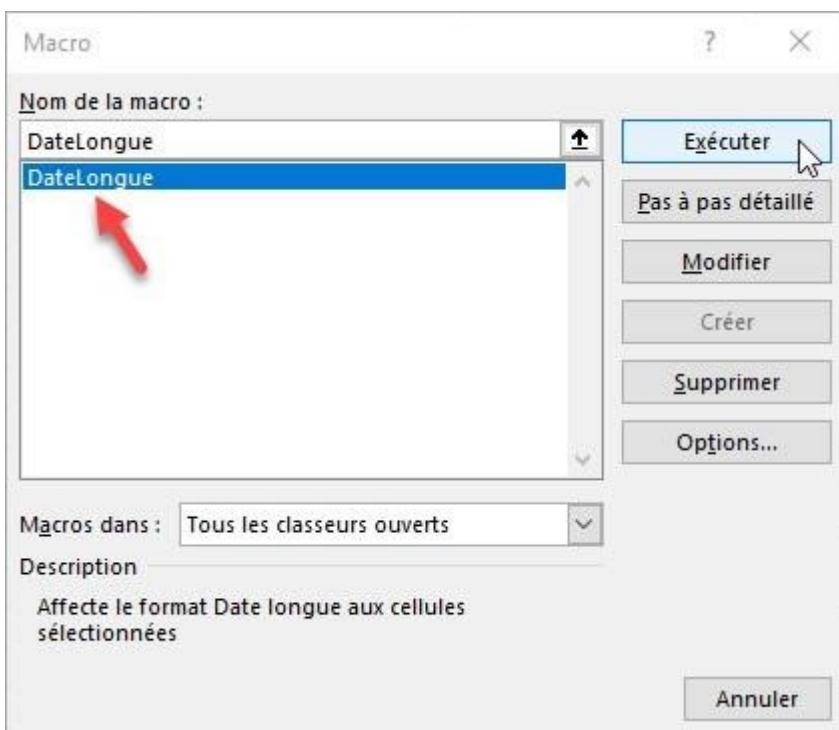
	A	B	C
1		26-mars	
2		17-janv	
3		15-sept	
4		18-févr	
5		12-juin	
6			
7			

Sélectionnez les cellules que vous venez de définir puis appuyez simultanément sur les touches *Contrôle*, *Majuscule* et *D* du clavier. La mise en forme est immédiate :

A	B	C
1	mercredi 26 mars 2025	
2	vendredi 17 janvier 2025	
3	lundi 15 septembre 2025	
4	mardi 18 février 2025	
5	jeudi 12 juin 2025	
6		
7		

Remarque

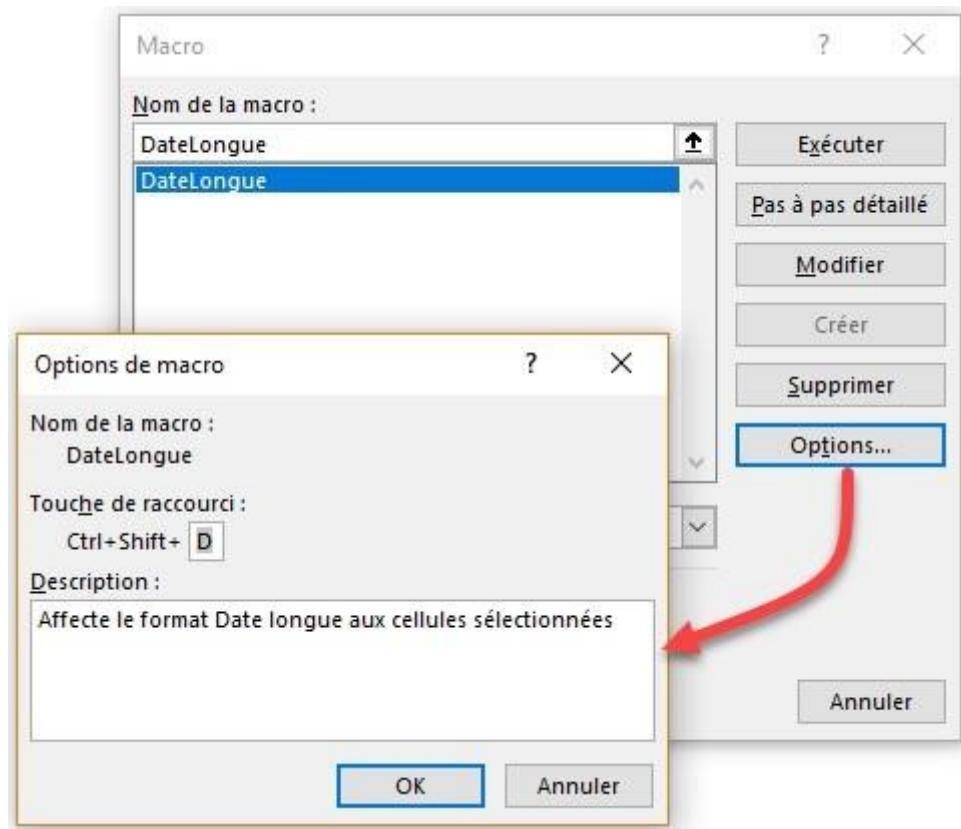
Pour exécuter la macro, vous pouvez également basculer sur l'onglet **Développeur** et cliquer sur l'icône **Macros** du groupe **Code**. La boîte de dialogue **Macro** s'affiche. Sélectionnez la macro à exécuter dans la zone d liste **Nom de la macro** et cliquez sur **Exécuter** :



Modifier une macro

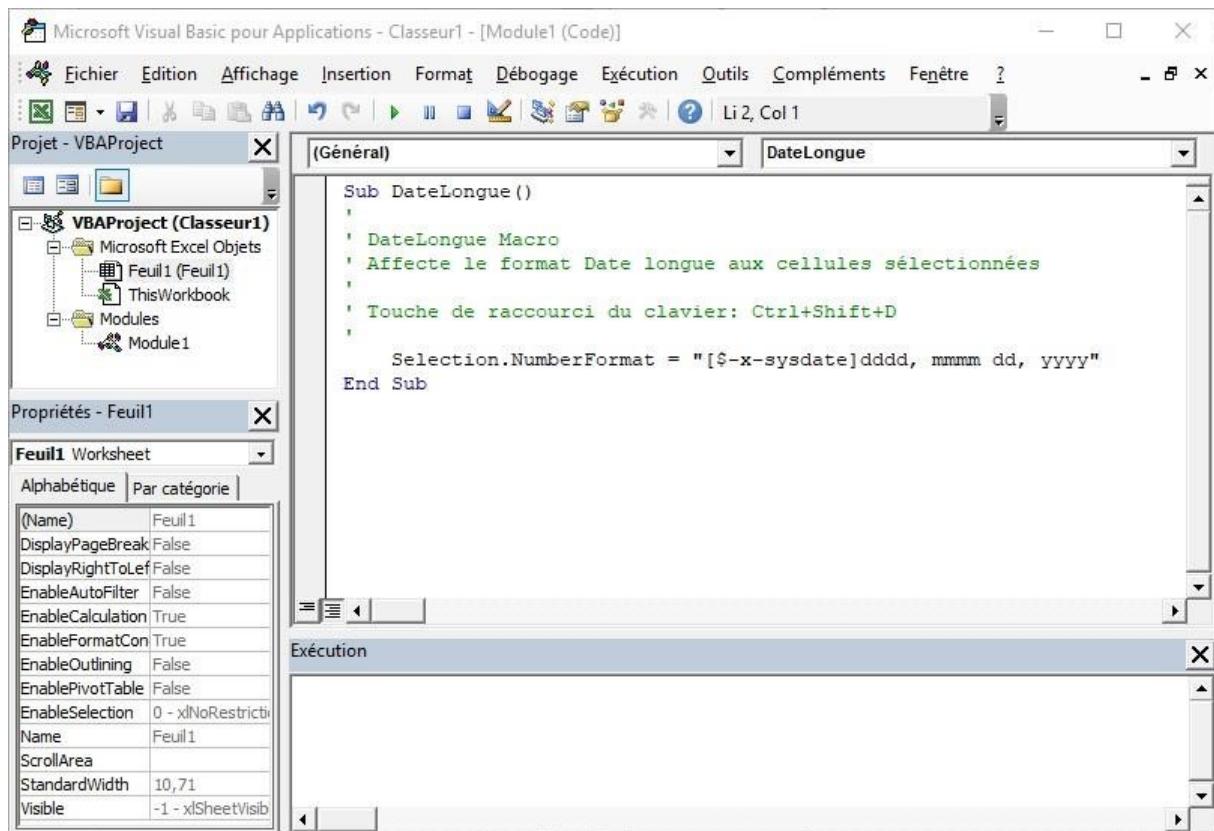
Une fois qu'une macro a été définie, vous pouvez toujours modifier ou définir son raccourci clavier ou sa définition. Cliquez sur l'icône **Macros**, dans le groupe **Code** de l'onglet **Développeur** du ruban. La boîte de dialogue **Macro** s'affiche. Cliquez sur la macro concernée puis cliquez sur **Options**. Une nouvelle boîte de

dialogue s'affiche dans laquelle vous pouvez définir ou modifier le raccourci clavier et la définition de la macro :



Faites les modifications nécessaires puis cliquez sur **OK** pour les prendre en compte.

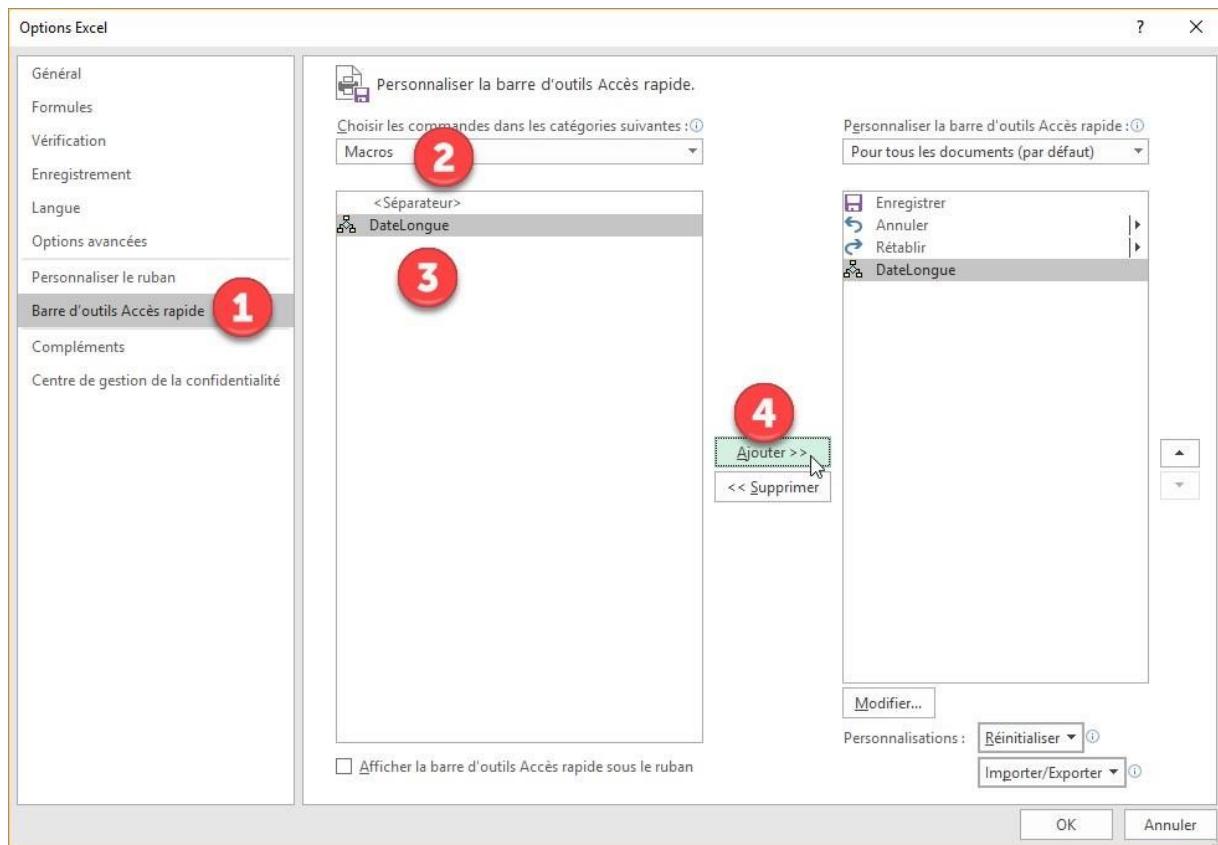
De retour dans la boîte de dialogue **Macro**, vous avez peut-être été tenté de cliquer sur le bouton **Modifier**. Cette action déclenche l'ouverture de la fenêtre **Microsoft Visual Basic pour Applications** dans laquelle vous voyez le code VBA généré par l'enregistreur de macros :



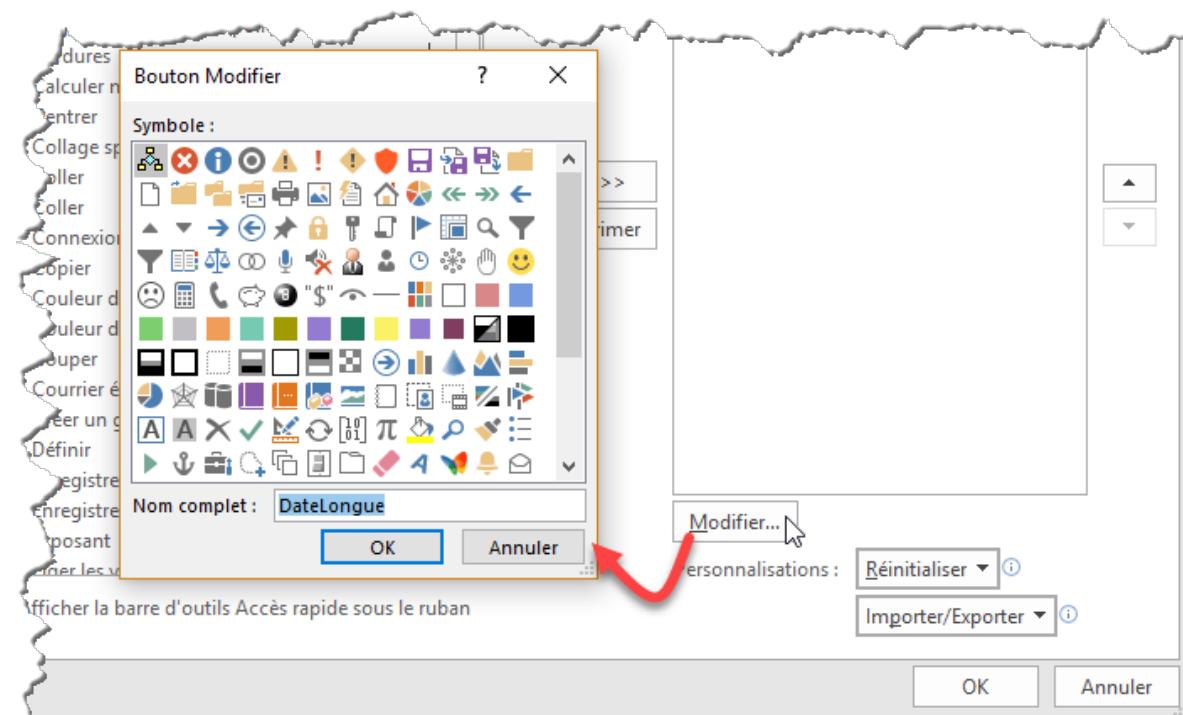
Chaque fois que vous créez une macro, Excel écrit du code VBA à votre place. Il est donc tout à fait possible de faire du VBA sans ... faire du VBA !

Affecter une macro à un bouton dans la barre d'outils Accès rapide

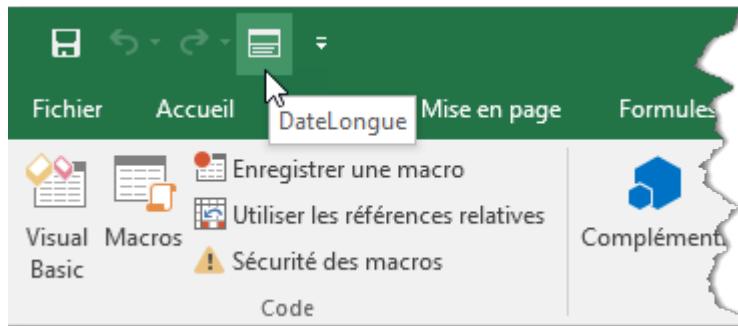
Pour accéder facilement à une macro, vous pouvez lui affecter une icône dans la barre d'outils **Accès rapide**. Lancez la commande **Options** dans le menu **Fichier**. La boîte de dialogue **Options Excel** s'affiche. Sélectionnez **Barre d'outils Accès rapide** dans la partie gauche de la boîte de dialogue (1). Sélectionnez **Macros** dans la liste déroulante **Choisir les commandes dans les catégories suivantes** (2). Cliquez sur la macro dans la zone de liste inférieure (3), puis cliquez sur **Ajouter** (4) :



Si l'icône par défaut ne vous convient pas, vous pouvez la changer en cliquant sur le bouton **Modifier**, dans la partie inférieure droite de la boîte de dialogue :



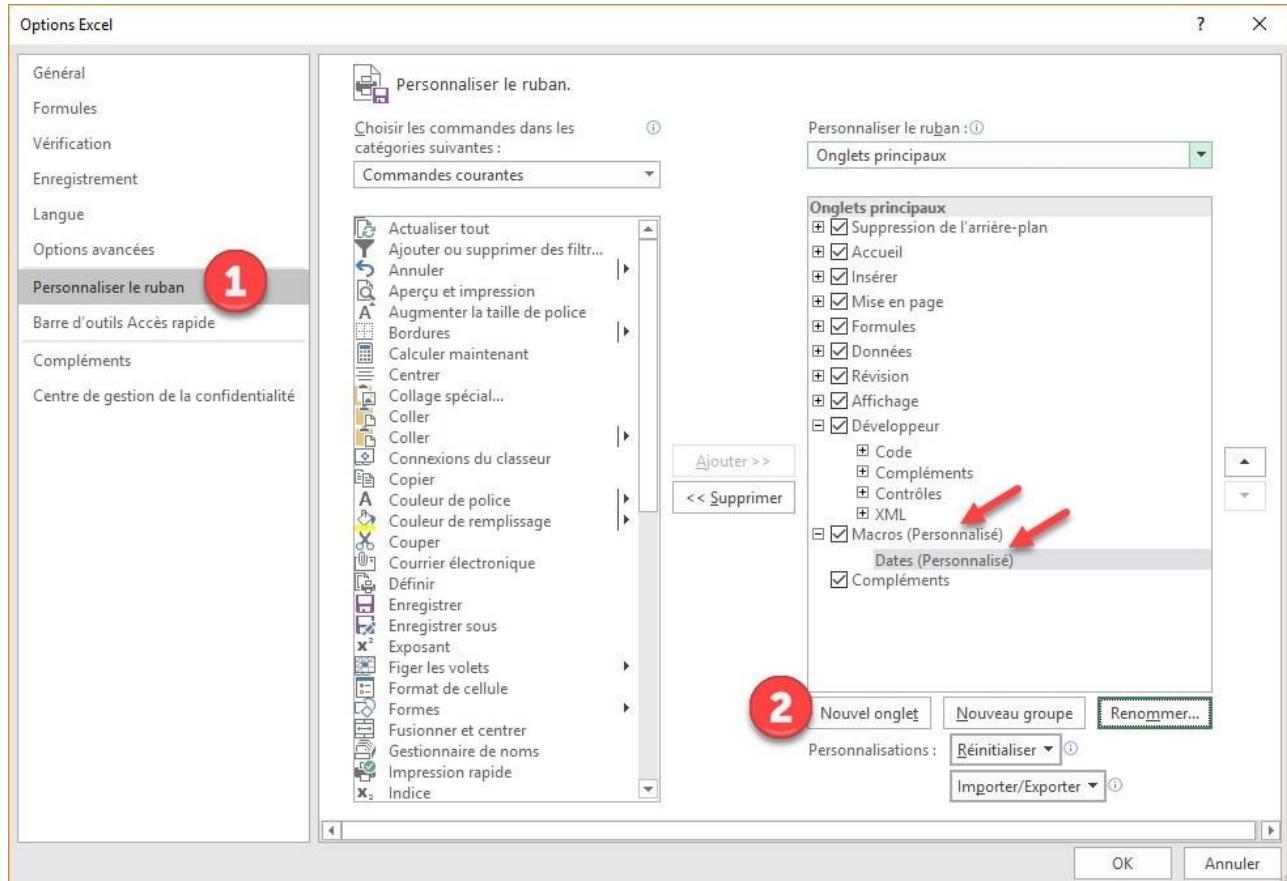
Choisissez une icône et cliquez sur **OK**. Il ne vous reste plus qu'à cliquer sur **OK** pour ajouter l'icône de la macro dans la barre d'outils **Accès rapide** :



Affecter une macro à une icône dans le ruban

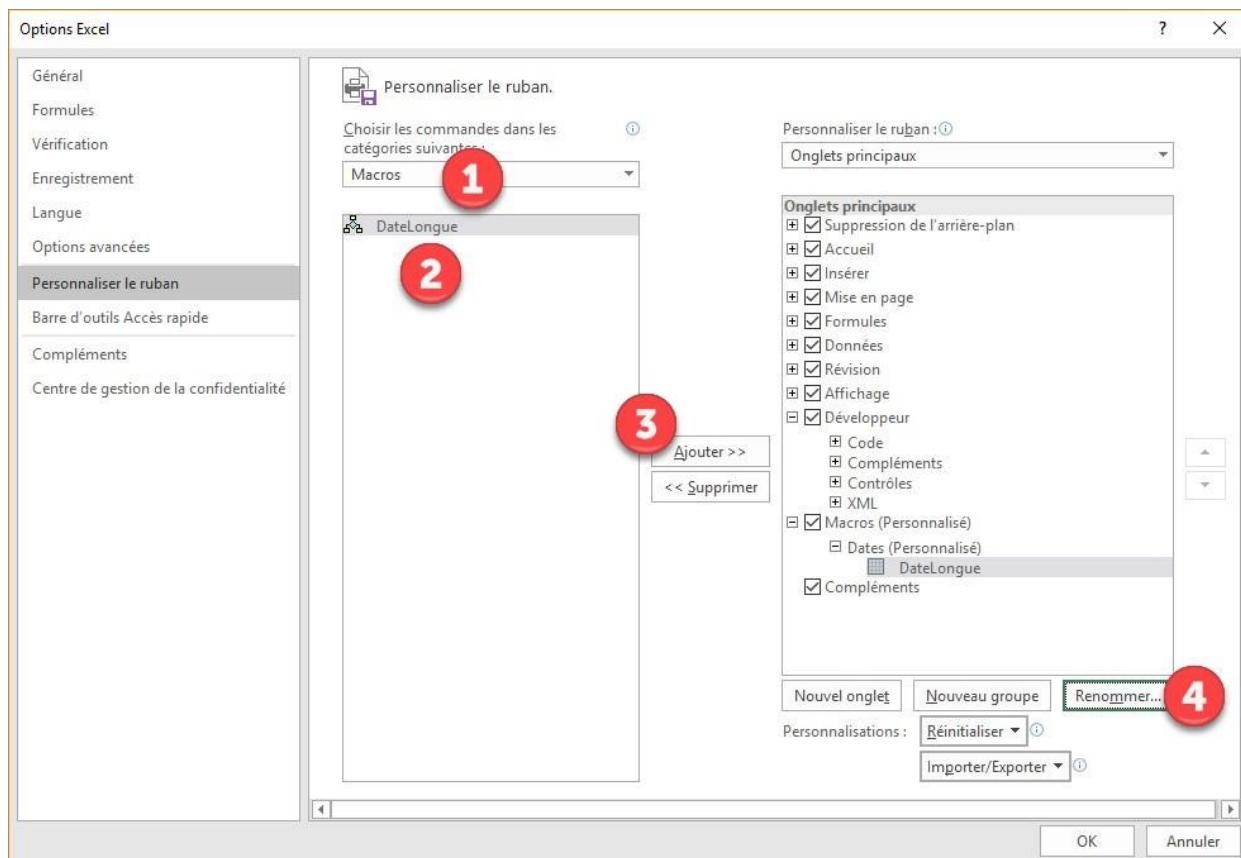
Si vous préférez, vous pouvez définir une icône dans le ruban pour exécuter la macro.

Lancez la commande **Options** dans le menu **Fichier**. La boîte de dialogue **Options Excel** s'affiche. Sélectionnez **Personnaliser le ruban** dans la partie gauche de la boîte de dialogue (1). Cliquez sur **Nouvel onglet** (2) pour définir un nouvel onglet dans le ruban. Cliquez sur cet onglet puis sur le bouton **Renommer** pour lui donner un nom plus approprié que « Nouvel onglet ». Ici par exemple, nous appellerons le nouvel onglet **Macros**. Cliquez sur **Nouveau groupe** puis sur **Renommer**. Donnez le nom **Dates** au nouveau groupe :



Il ne vous reste plus qu'à insérer une icône qui représente la macro dans le groupe **Dates**. Sélectionnez **Macros** (1) dans la liste déroulante. Assurez-vous que le groupe **Dates** est sélectionné dans la zone de liste de droite, cliquez sur **DateLongue** (2) puis sur **Ajouter** (3).

Si nécessaire, vous pouvez cliquer sur **Renommer** (4) pour choisir l'icône de la macro **DateLongue** :

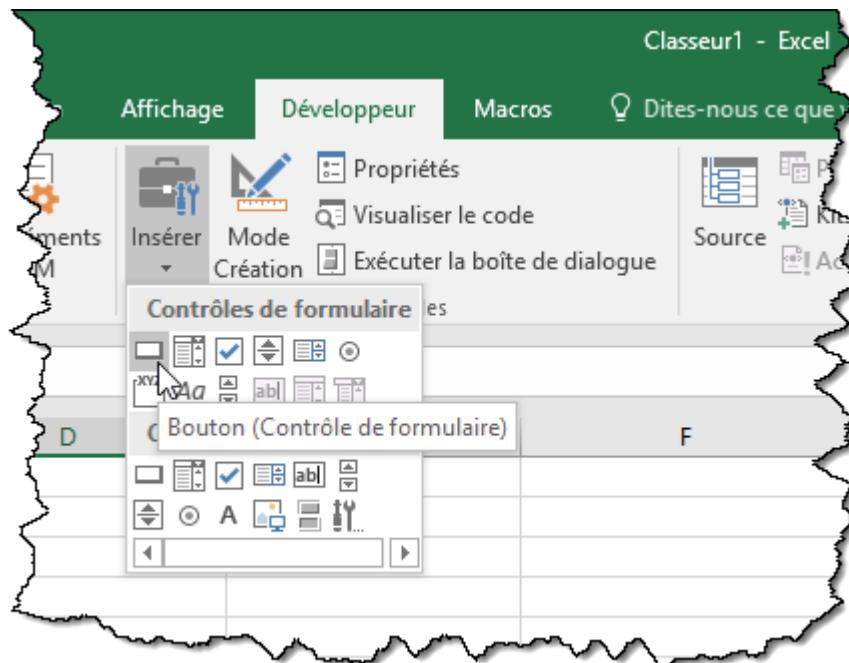


Fermez la boîte de dialogue **Options Excel** en cliquant sur **OK**. La macro **DateLongue** est accessible dans le groupe **Dates**, sous l'onglet **Macros** du ruban :

Affecter une macro à un bouton de formulaire

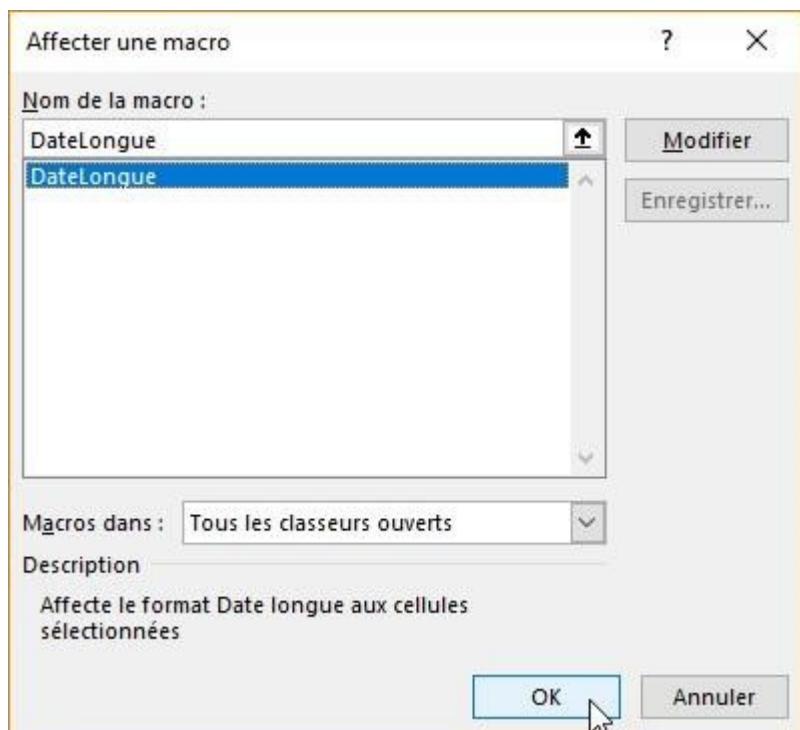
Pour terminer, nous allons voir comment affecter une macro à un bouton de formulaire.

Basculez sur l'onglet **Développeur**. Dans le groupe **Contrôles**, cliquez sur l'icône **Insérer**, puis sur l'icône **Bouton (Contrôle de formulaire)** :



Dessinez le bouton sur la feuille de calcul en maintenant le bouton gauche de la souris enfoncé. Au relâchement du bouton gauche, la boîte de dialogue **Affecter une macro** s'affiche. Sélectionnez la macro dans la liste et cliquez sur **OK** :



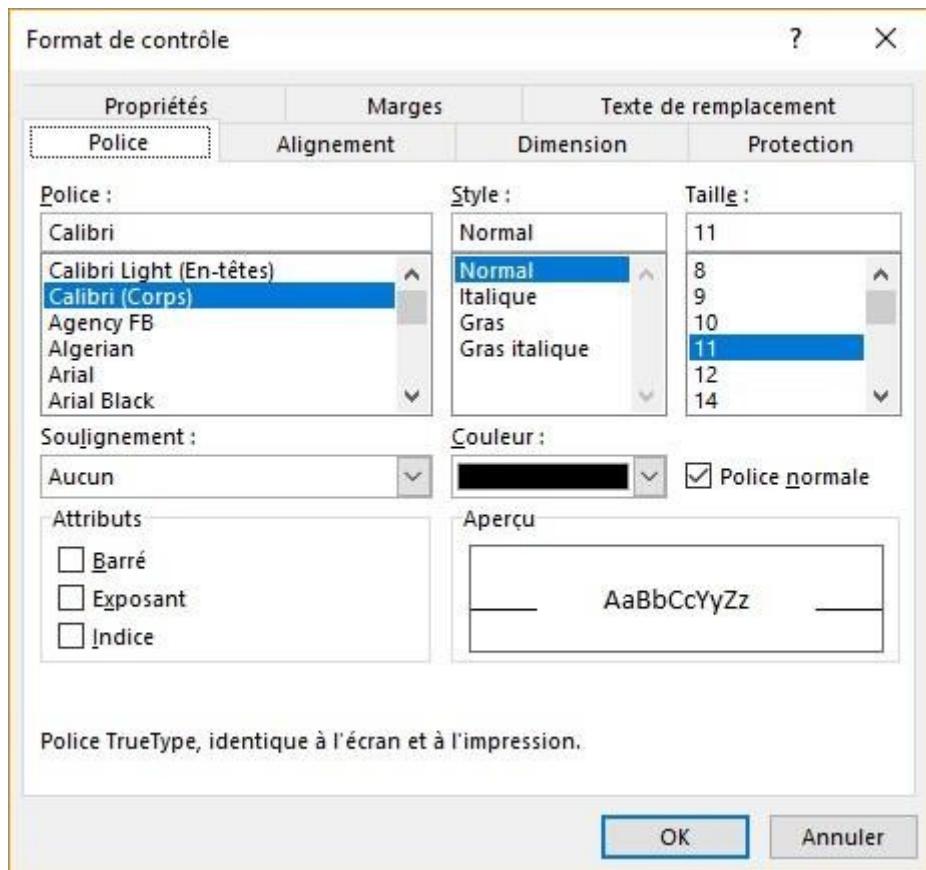


Cliquez sur le libellé du bouton et modifiez-le :

	A	B	C	D
1		mercredi 26 mars 2025		
2		vendredi 17 janvier 2025		
3		lundi 15 septembre 2025		
4		mardi 18 février 2025		
5		jeudi 12 juin 2025		Date longue
6				
7				

Si nécessaire, vous pouvez modifier la taille du bouton. Cliquez dessus et agissez sur ses poignées de redimensionnement. De même, vous pouvez modifier son emplacement. Pointez le, maintenez le bouton droit de la souris enfoncé et déplacez-le à l'endroit souhaité. Au relâchement du bouton droit de la souris, sélectionnez **Placer ici** dans le menu.

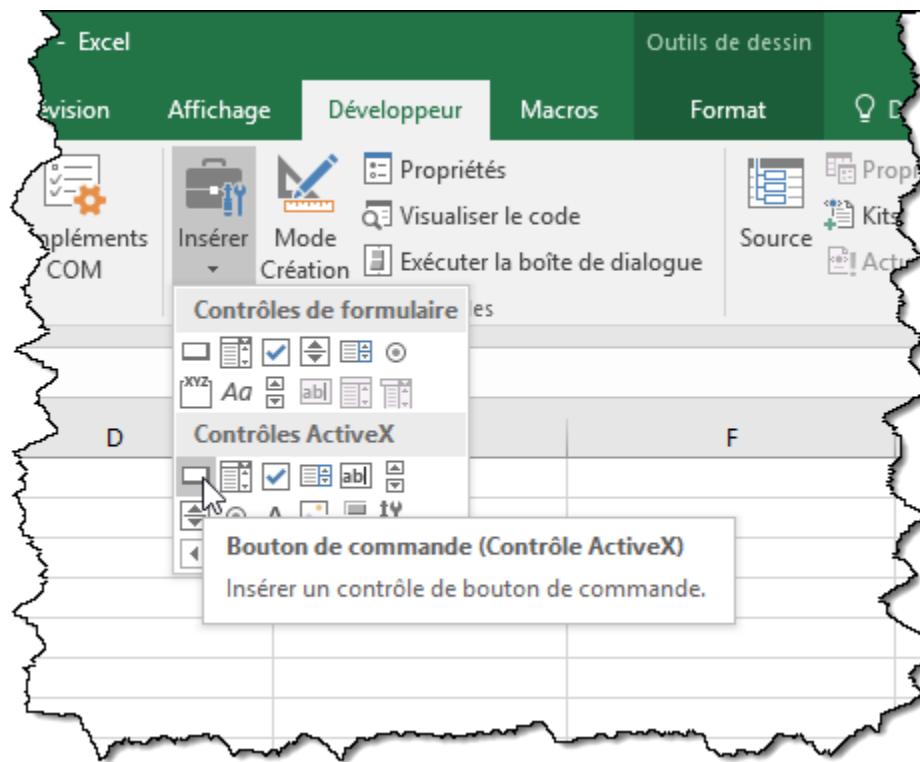
Les boutons de formulaire sont assez peu personnalisables. Pour accéder aux options disponibles, cliquez du bouton droit sur le bouton et sélectionnez **Format de contrôle** dans le menu :



Affecter une macro à un bouton de commande

Si vous voulez un bouton personnalisable, vous utiliserez un bouton de commande et non un bouton de formulaire.

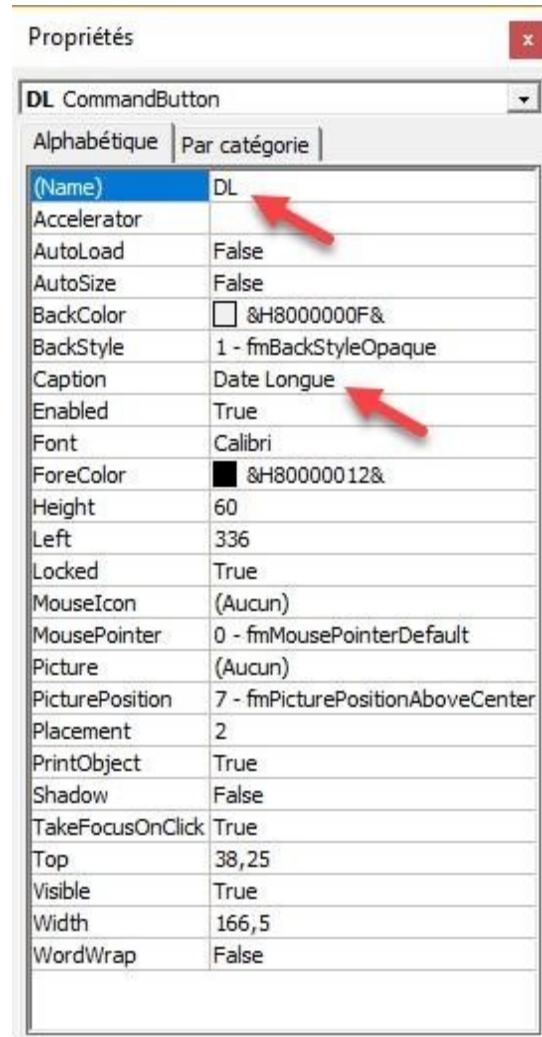
Basculez sur l'onglet **Développeur**. Dans le groupe **Contrôles**, cliquez sur l'icône **Insérer**, puis sur l'icône **Bouton de commande (Contrôle ActiveX)** :



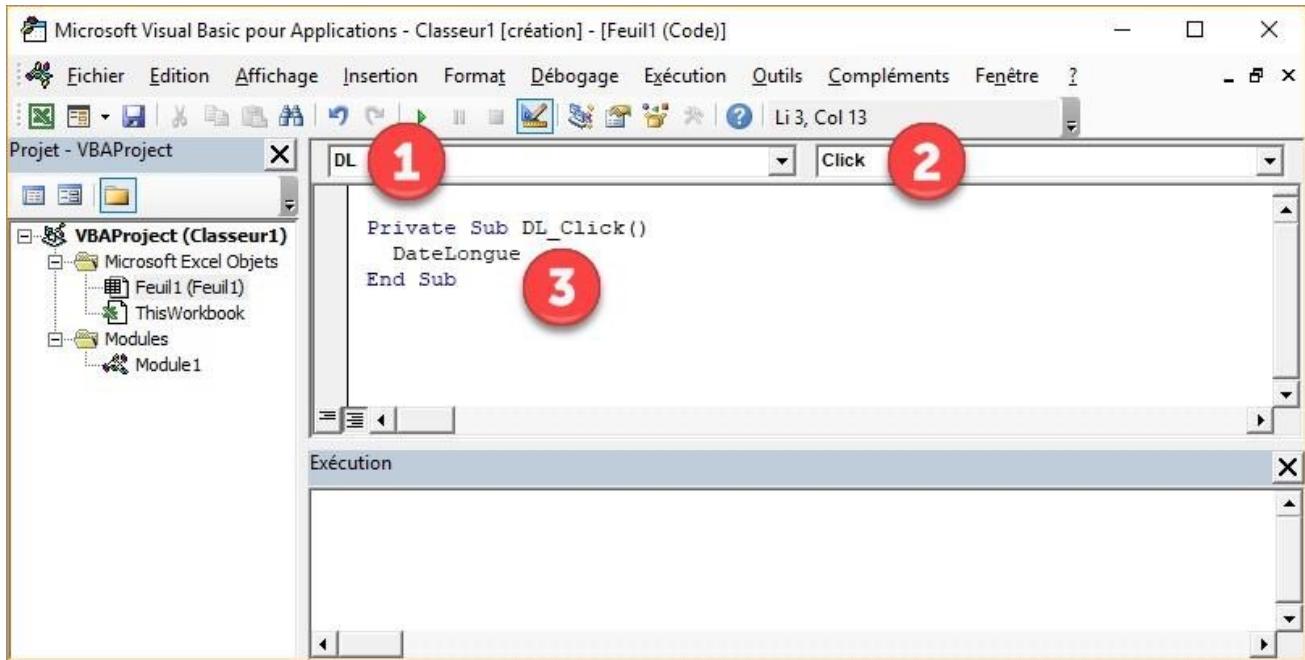
Dessinez le bouton sur la feuille de calcul en maintenant le bouton gauche de la souris enfoncé. Au relâchement du bouton gauche, un bouton intitulé **CommandButton** s'affiche.



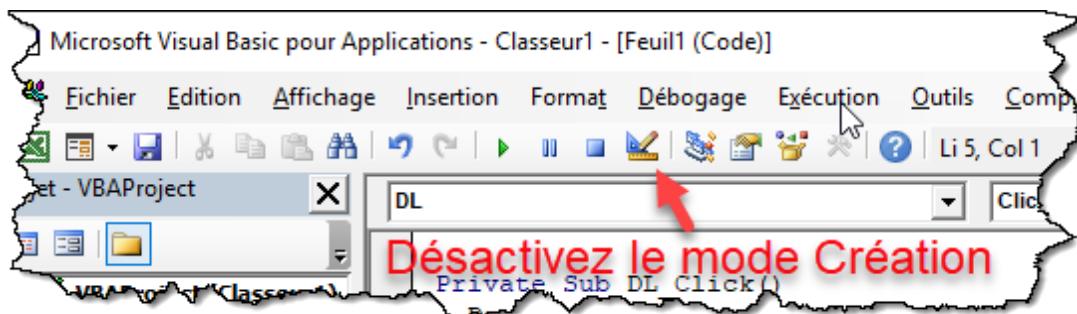
Cliquez du bouton droit sur le bouton de commande et sélectionnez **Propriétés** dans le menu. La boîte de dialogue **Propriétés** s'affiche. Utilisez les propriétés **(Name)** et **Caption** pour respectivement donner un nom au bouton de commande (ici, **DL**) et choisir son libellé (ici, **Date Longue**) :



Sous l'onglet **Développeur**, dans le groupe **Contrôles**, cliquez sur **Visualiser le code**. Cette action affiche la fenêtre **Microsoft Visual Basic pour Applications**. Sélectionnez **DL** dans la première liste déroulante (1) et **Click** dans la seconde (3). Un code VBA s'affiche dans la partie centrale de la fenêtre. Entrez le nom de la macro (ici **DateLongue**) puis fermez la fenêtre **Microsoft Visual Basic pour Applications** :

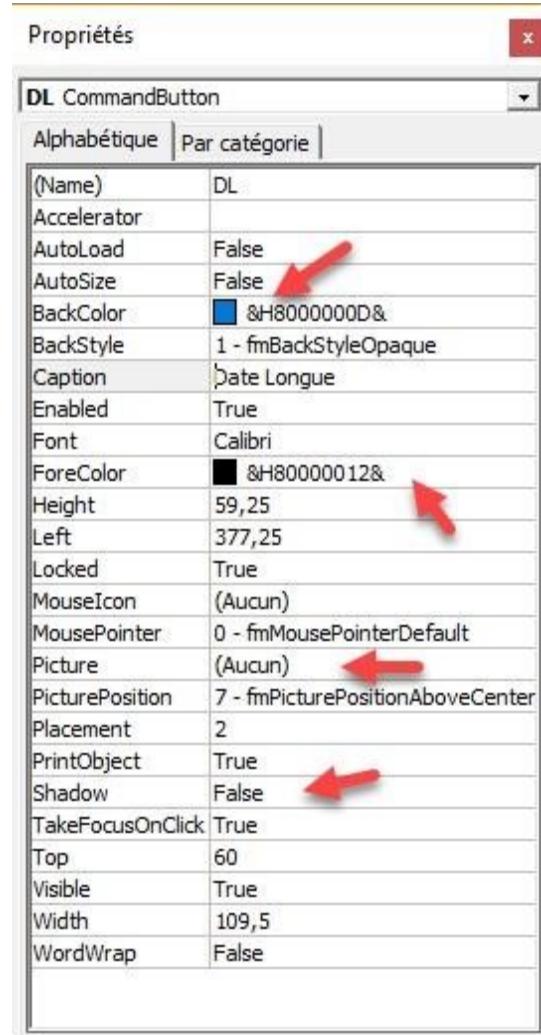


Cliquez sur l'icône **Mode Création** pour quitter ce mode de fonctionnement :



Vous pouvez vérifier que le bouton de commande fonctionne en sélectionnant les cellules à mettre en forme et en cliquant sur le bouton.

Pour terminer, voyons comment personnaliser le bouton. Cliquez sur l'icône **Mode Création** (onglet **Développeur**, groupe **Contrôles**) pour passer en mode **Création**. Cliquez sur le bouton puis sur l'icône **Propriétés**. Vous pouvez choisir entre autres la couleur d'arrière-plan et la couleur d'écriture du bouton, l'image d'arrière-plan du bouton et son ombrage :



Une fois le bouton personnalisé, cliquez sur l’icône **Mode Création** (onglet **Développeur**, groupe **Contrôles**) pour pouvoir l’utiliser.

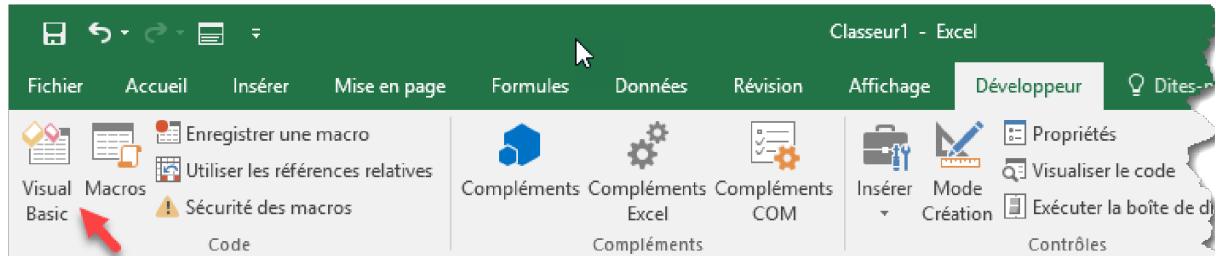
Vous en savez maintenant assez sur l’enregistreur de macros pour créer vos propres macros et les exécuter :

- depuis la boîte de dialogue **Macros**;
- avec un raccourci clavier ;
- en cliquant sur une icône dans la barre d’outils **Lancement rapide**;
- en cliquant sur une icône dans le ruban ;
- en cliquant sur un bouton de contrôle ou un bouton de commande.

L’application Microsoft Visual Basic pour Applications

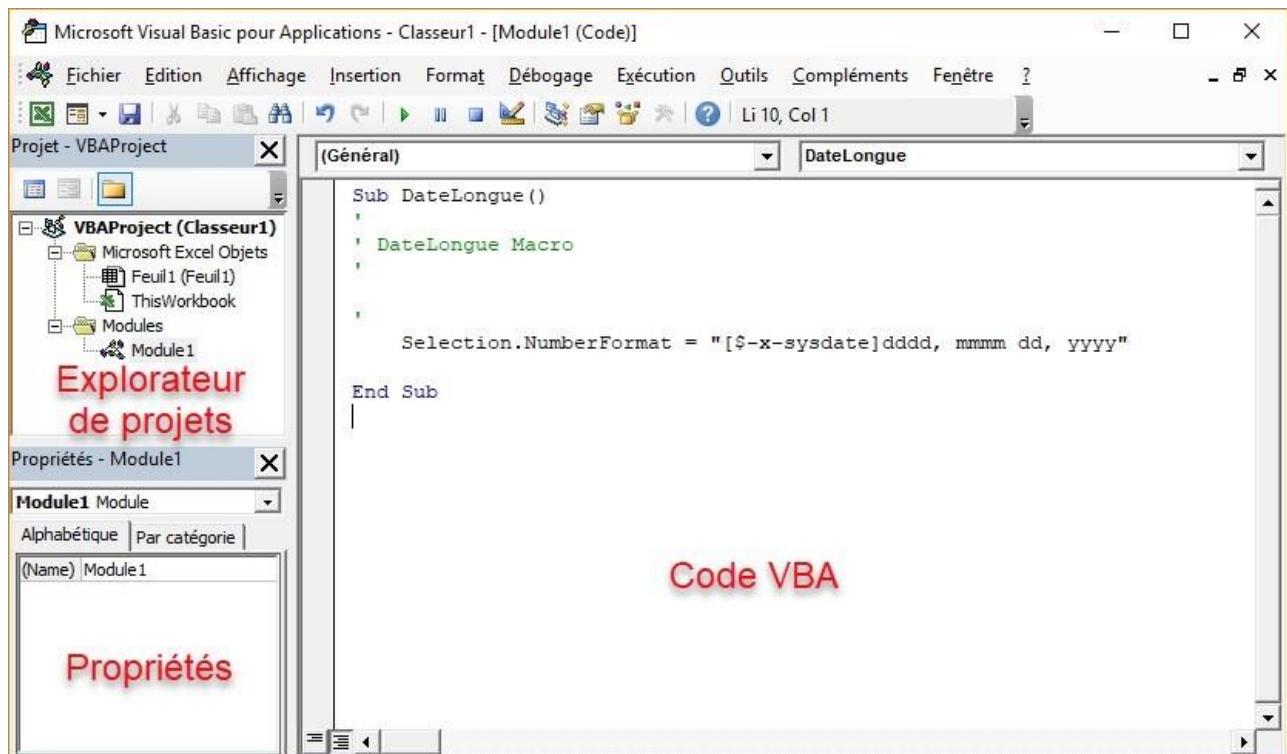
Dans cette section, vous allez faire connaissance avec l’application Microsoft Visual Basic for Applications, dans laquelle vous développerez vos projets VBA.

Pour accéder à cette fenêtre, basculez sur l'onglet **Développeur** du ruban et cliquez sur l'icône **Visual Basic** dans le groupe **Code** :



Si vous n'êtes pas réfractaire aux raccourcis clavier, vous pouvez également appuyer sur *Alt + F11* pour parvenir au même résultat.

Exampons la fenêtre **Microsoft Visual Basic pour Applications** :



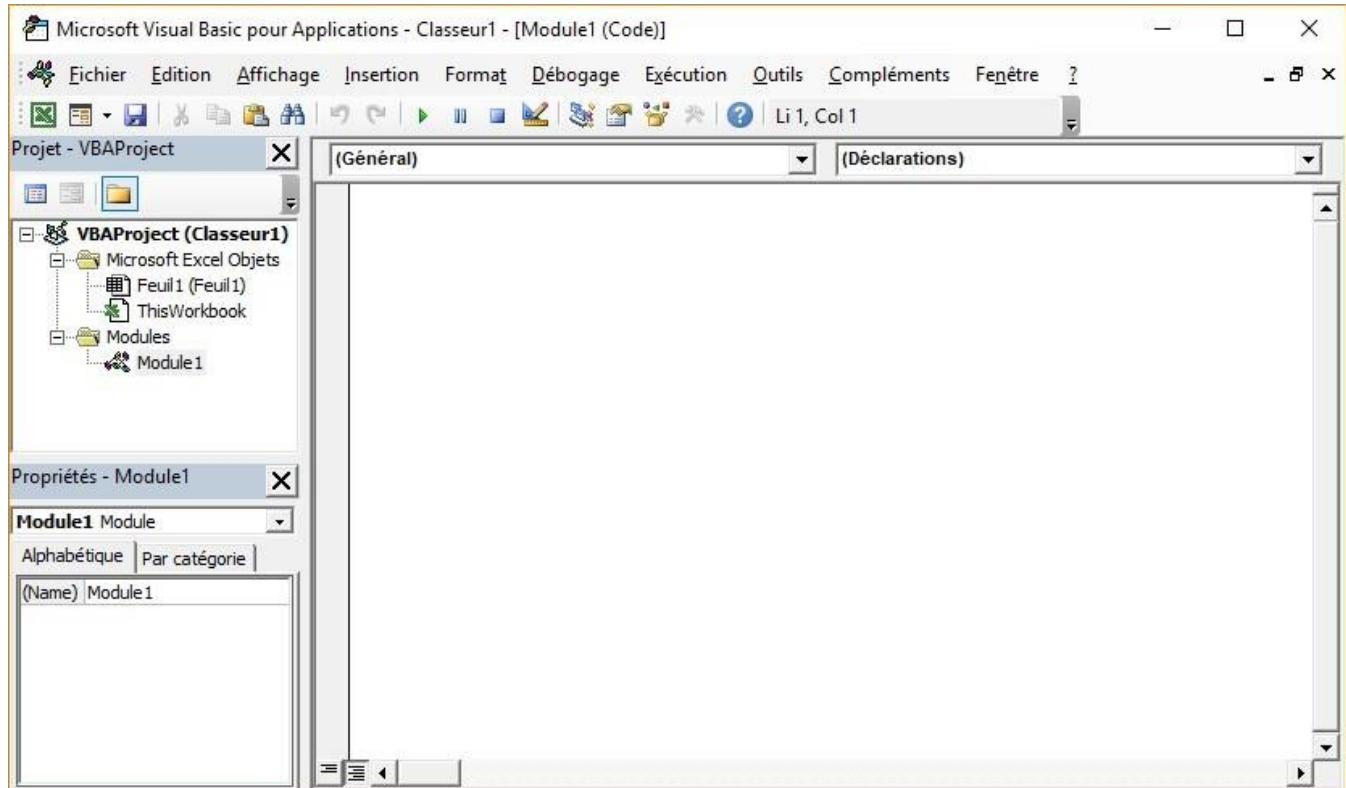
Il se peut que votre fenêtre soit légèrement différente. Les volets en trop ou manquants peuvent être affichés/supprimés avec les commandes du menu **Affichage**.

Premiers pas en VBA

Examinez le volet **Explorateur de projets**.

Si vous n'avez pas encore défini de macros dans le classeur en cours, vous allez créer un module. C'est en effet dans ce module que les instructions VBA rattachées au classeur en cours seront définies. Lancez la commande **Module** dans le menu **Insertion**. Le dossier **Modules** et l'entrée **Module1** sont ajoutés dans

l'explorateur de projets et une feuille blanche apparaît dans la partie droite de la fenêtre :



Si vous avez défini une ou plusieurs macros, le dossier **Modules** et l'entrée **Module1** doivent apparaître dans ce volet.

Supposons que vous ayez défini la macro **DateLongue**, comme indiqué dans la section « Une première macro ». Vous devriez avoir le code suivant dans le module 1 :

```
Sub DateLongue()  
  
'  
  
' DateLongue Macro  
  
'  
  
    Selection.NumberFormat = "[-$x-sysdate]dddd, mmmm dd, yyyy"  
End Sub
```

Sur la première ligne, vous retrouvez le nom de la macro **DateLongue**, précédé du mot **Sub**, pour **subroutine**, ou **procédure** en français. La procédure se termine par les mots **End Sub**. Le code de la procédure **DateLongue** se trouve entre les mots **Sub** et **End Sub**.

Dans cet exemple précis, vous trouvez plusieurs lignes de commentaires, qui commencent par une apostrophe :

```
' DateLongue Macro
```

```
'
```

Ainsi qu'une ligne qui vous laisse peut-être perplexe :

```
Selection.NumberFormat = "[-$x-sysdate]dddd, mmmm dd,  
yyyy"
```

Il s'agit d'une notation objet, car oui, VBA est un langage objet !

Si vous n'avez aucune idée de ce qu'est un langage objet, cliquez sur ce lien (renvoie vers la section « Si vous n'avez aucune idée de ce qu'est un langage objet »).

Cette ligne de code définit la propriété **NumberFormat** de l'objet **Selection**. En d'autres termes, le format des cellules sélectionnées. La chaîne affectée à cette propriété a été générée par l'enregistreur de macros. Elle indique que les dates doivent être affichées au format long :

- dddd : nom du jour au format long
- mmmm : nom du mois au format long
- dd : numéro du jour
- yyyy : année au format long

La première partie du format ([-\$x-sysdate]) indique que la date sera affichée en fonction de la langue système. Ici, il s'agit du français. La date 25/03/2025 sera transformée en mercredi 25 mars 2025.

Si vous n'avez aucune idée de ce qu'est un langage objet

Un petit aparté pour ceux qui ne savent pas ce qu'est un langage objet et/ou qui n'ont jamais programmé en objet.

Eh bien, comme son nom l'indique, un langage objet manipule ... des objets !

Vous pouvez considérer un objet comme une boîte qui possède des propriétés et des méthodes. Les propriétés définissent les caractéristiques de l'objet et les méthodes agissent sur l'objet. Si nous prenons le cas particulier d'Excel, Les programmes VBA manipulent un ensemble d'objets mis à disposition du programmeur par Excel : des cellules, des lignes, des colonnes, des plages sélectionnées, des feuilles de calcul et des classeurs.

Propriétés

Pour accéder à une propriété, vous écrirez quelque chose comme ceci :

```
Objet.Property
```

Où **Objet** est le nom de l'objet et **Propriété** est la propriété à laquelle vous voulez accéder.

Vous pouvez lire la valeur d'une propriété et l'afficher dans une boîte de dialogue avec une instruction **Msg**. Par exemple :

```
Msg Objet.Property
```

Ou encore affecter une valeur à une propriété avec un simple signe = (égal à).

Si la valeur est numérique, il suffit de l'indiquer après le signe = :

```
Objet.Property = Valeur
```

Si la valeur est une chaîne de caractères, elle sera encadrée par des guillemets :

```
Objet.Property = "Valeur"
```

Méthodes

Pour appliquer une méthode à un objet, il suffit d'indiquer le nom de l'objet, suivi d'un point, suivi du nom de la méthode. Par exemple, pour basculer sur la feuille de calcul **Feuil2** du classeur courant, vous appliquerez la méthode **Activate** à l'objet **Worksheets(« Feuil2 »)** :

```
Worksheets("Feuil2").Activate
```

Ou encore, pour sélectionner la cellule **B5** dans la feuille courante, vous appliquerez la méthode **Select** à l'objet **Range(« B5 »)** :

```
Range("B5").Select
```

Et maintenant, tout le travail va consister à connaître les objets d'Excel, leurs propriétés et leurs méthodes. Vous voyez que ce n'est pas si compliqué que ça !

Procédures

Lorsque vous définissez une macro avec l'enregistreur de macros, Excel crée une procédure, lui donne le nom de la macro et la stocke dans le module attaché au classeur. Nous allons vérifier que l'inverse est également vrai. En d'autres termes, que si vous définissez une procédure en VBA, elle est accessible sous la forme d'une macro.

Une première procédure

A titre d'exemple, vous allez utiliser le code suivant dans le module attaché au classeur :

```
Sub EnRouge ()  
    '  
    ' Arrière-plan rouge  
    '  
    Selection.Interior.Color = RGB(255, 0, 0)  
End Sub
```

La procédure s'appelle **EnRouge**. Elle utilise la fonction **RGB()** pour affecter la couleur rouge (**RGB(255,0,0)**) à l'arrière-plan (**Interior.Color**) des cellules sélectionnées (**Selection**).

Vous voyez, il n'y a rien de bien compliqué. Le tout est de connaître les termes à utiliser et de les utiliser dans le bon ordre.

La fonction **RGB()** ne vous dit peut-être rien. Dans ce cas, sachez qu'il s'agit d'une fonction qui se retrouve dans la plupart des langages de programmation. Elle définit une couleur par ses composantes **Red** (rouge), **Green** (vert) et **Blue** (bleu).

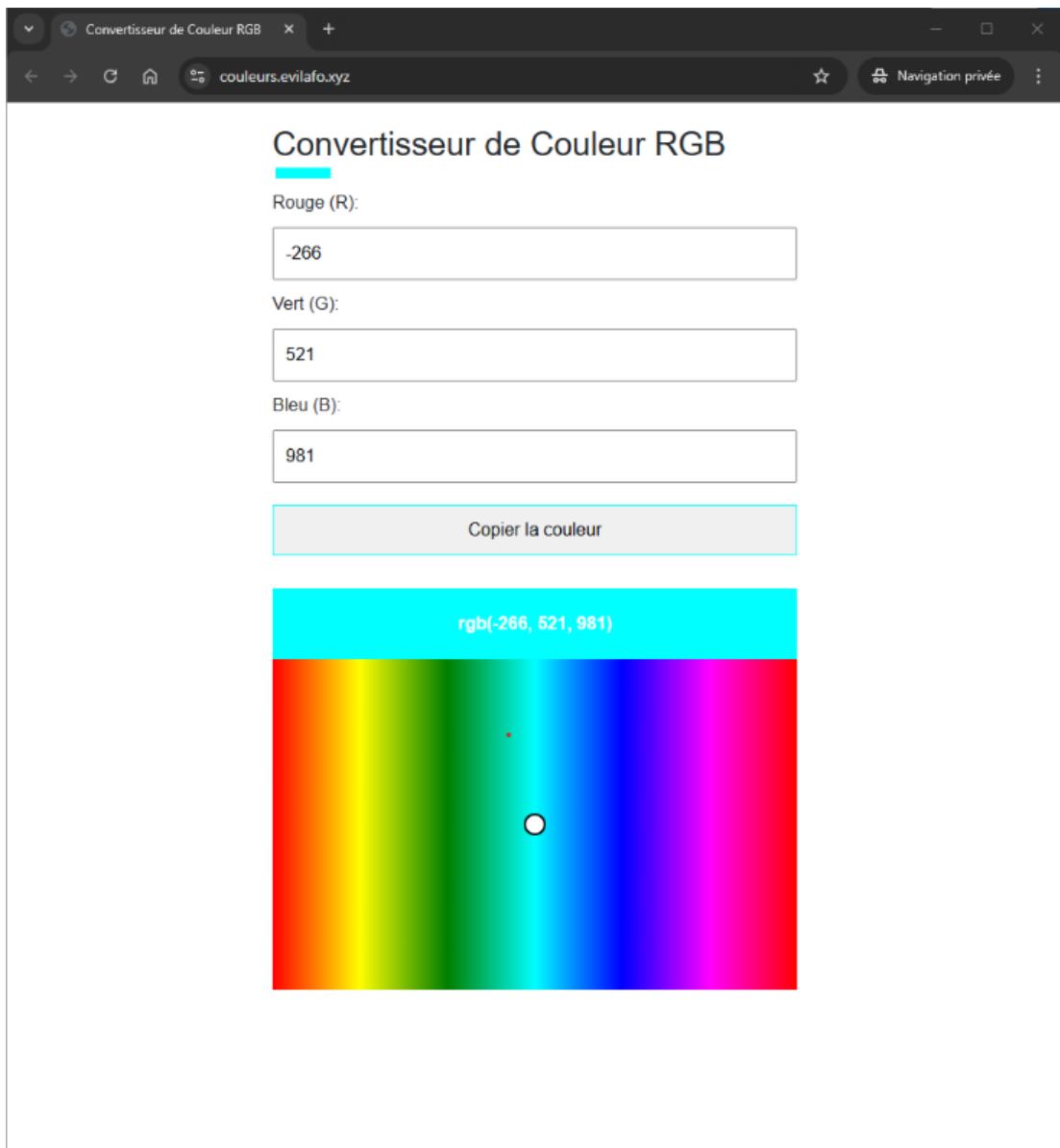
La « force » de chaque composante est donnée par un nombre entier compris entre 0 et 255. Si une composante vaut 0, la couleur correspondante n'est pas du tout présente. Inversement, si une composante vaut 255, la couleur correspondante est présente à 100%.

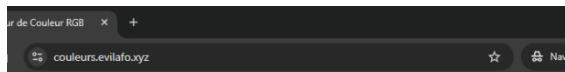
En extrapolant, il est facile de comprendre que vous disposez de 256 niveaux de rouge, de 256 niveaux de vert et de 256 niveaux de bleu. Ce qui représente $256 \times 256 \times 256 = 16\,777\,216$ couleurs. Un peu plus de 16 millions de couleurs : il y a de quoi faire !

Rassurez-vous, il n'est pas nécessaire de connaître toutes les valeurs des composantes RVB. Pour définir une couleur, vous utiliserez une application graphique quelconque : **couleurs.evilaf0** par exemple sur le web :

<https://couleurs.evilaf0.xyz>

Lancez **couleurs.evilaf0**, cliquez sur la couleur dans la palette d'outils, déplacez le signe **Plus** dans la palette et cliquez sur « copier la couleur » pour copier. Les composantes RGB sont disponibles dans les cases **Rouge**, **Vert** et **Bleu** :





Convertisseur de Couleur RGB

Rouge (R):

317

Vert (G):

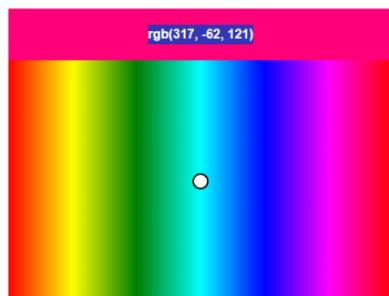
-62

Bleu (B):

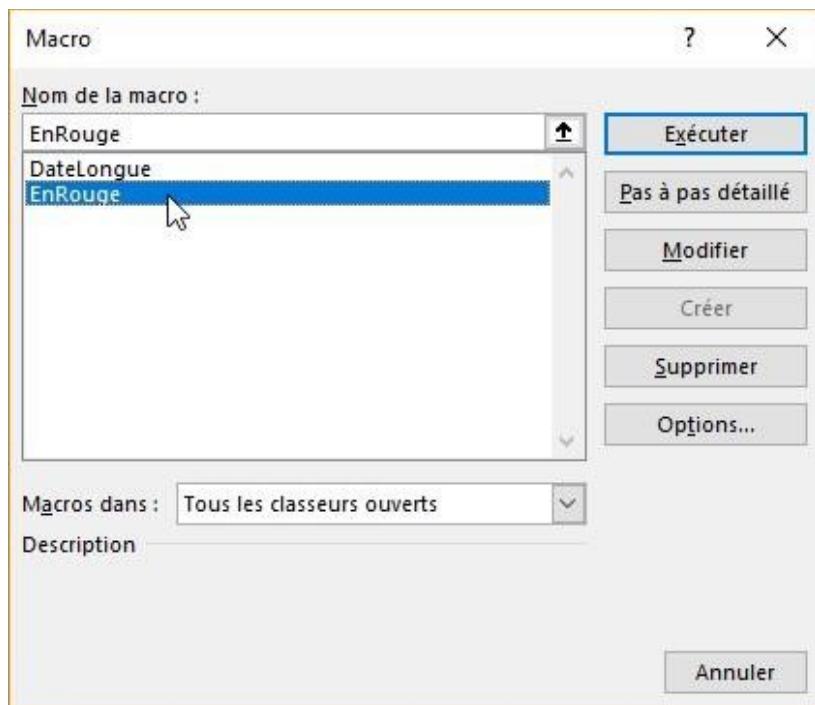
121

Texte copié dans le presse-papiers !

Copier la couleur

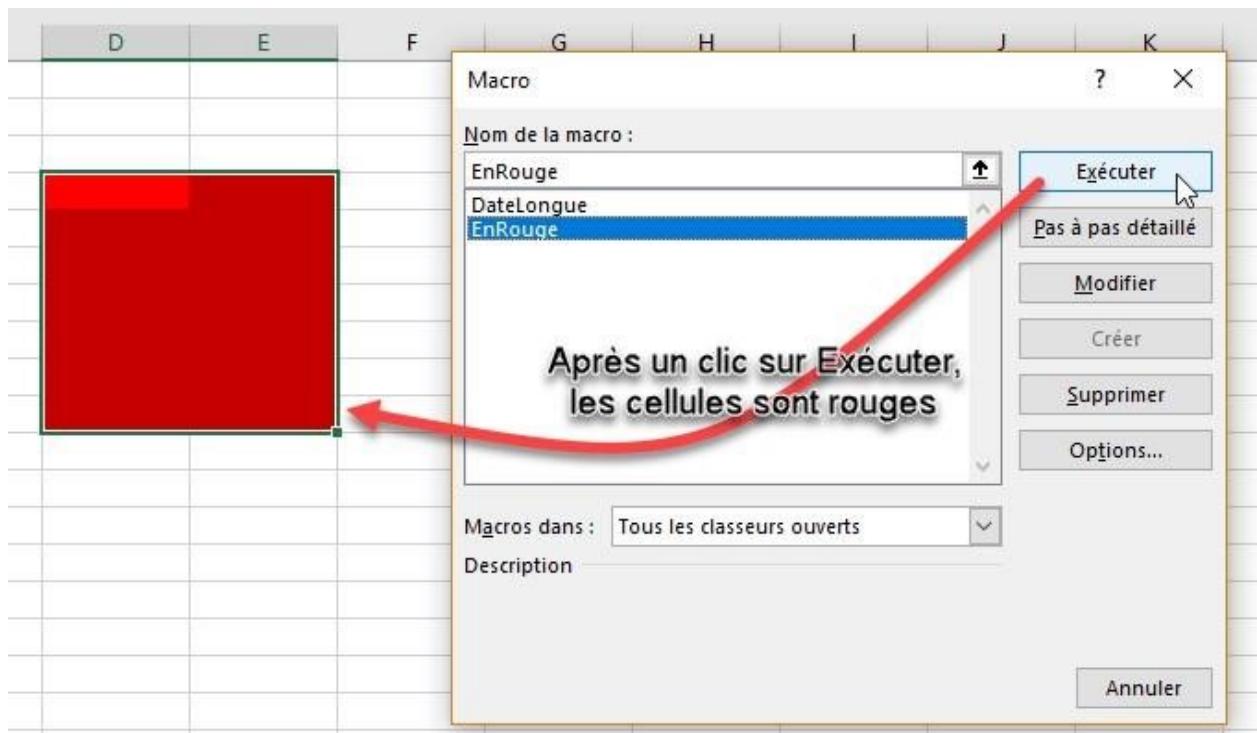


Voyons maintenant si la procédure **EnRouge** est bien disponible sous la forme d'une macro. Il n'y a rien de plus simple. Basculez sur l'onglet **Développeur** dans le ruban, puis cliquez sur l'icône **Macros** dans le volet **Code**. Comme vous le voyez, la macro **EnRouge** a bien été créée :



Fermez la boîte de dialogue **Macro**.

Pour voir si le code fonctionne, sélectionnez plusieurs cellules dans la feuille. Toujours sous l'onglet **Développeur** du ruban, cliquez sur **Macros**, dans le groupe **Code**, sélectionnez la macro **EnRouge** et cliquez sur **Exécuter**. Voici le résultat :



Avant de terminer cette section, j'ai une question pour vous. Vous avez vu que cette instruction affectait un arrière-plan rouge aux cellules sélectionnées :

```
Selection.Interior.Color = RGB(255, 0, 0)
```

Quelle instruction devriez-vous utiliser pour affecter la couleur verte aux caractères qui se trouvent dans les cellules sélectionnées ?

```
Selection.Font.Color = RGB(0, 255, 0)
```

Pour terminer, changez le nom et le commentaire de la procédure **EnRouge()** :

```
Sub EnVert()
    '
    ' Caractères en vert
    '
    Selection.Font.Color = RGB(0, 255, 0)
End Sub
```

Fonctions

Les fonctions sont définies par le mot-clé **Function** :

```

Function Nom(Param1 as Type1, Param2 as Type2, ... ParamN as TypeN)

    ' Une ou plusieurs instructions

    Nom = Valeur

End Function

```

Ici :

- **Nom** est le nom de la fonction ;
- **Param1 à ParamN** sont les paramètres passés à la fonction ;
- **Type1 à TypeN** sont les types des paramètres **Param1 à ParamN**;
- **Valeur** est la valeur qui doit être retournée par la fonction. S'il s'agit d'une valeur chaîne, pensez à l'entourer par des guillemets.

Pour l'instant, nous allons nous intéresser au type **Integer**. Ce type caractérise les nombres entiers compris entre **-32 768** et **32 767**. Dans une section à venir, vous apprendrez quels sont les types utilisables en VBA.

Pour prendre un exemple très simple, nous allons définir la fonction **Surface()** qui calcule la surface d'un rectangle à partir de ses deux arguments : la longueur et la hauteur du rectangle.

```
Function Surface(longueur As Integer, hauteur As Integer)
```

```
    Surface = longueur * hauteur
```

```
End Function
```

Nous allons supposer que la longueur et la hauteur du rectangle se trouvent dans les cellules **A2** et **B2** :

	A	B	C	D
1	Longueur du rectangle	Hauteur du rectangle	Surface du rectangle	
2	14	28		
3				
4				
5				
6				

Pour afficher la surface du rectangle dans la cellule **C2**, nous allons définir la procédure **CalculSurface()** :

```

Sub CalculSurface()

    Range("C2").Value = Surface(Range("A2").Value, Range("B2").Value)

End Sub

```

Ce code est très simple. La valeur des cellules **A2** et **B2** est lue dans la propriété **Value** des objets **Range(« A2 »)** et **Range(« B2 »)**. Ces deux valeurs sont passées en argument de la fonction **Surface()**. Le résultat retourné par la fonction **Surface()** est copié dans la cellule **C2** à l'aide de la propriété **Value** de l'objet **Range(« C2 »)**.

Voici le résultat :

The screenshot shows a Microsoft Excel spreadsheet with a table containing three columns: 'Longueur du rectangle', 'Hauteur du rectangle', and 'Surface du rectangle'. The values in the first two columns are 14 and 28 respectively, and the value in the third column is 392. A red arrow points from the cell containing 392 to the 'Macro' dialog box. The dialog box is titled 'Macro' and contains a list of macros: 'CalculSurface', 'CalculSurface', 'DateLongue', and 'EnRouge'. The second item in the list, 'CalculSurface', is selected. On the right side of the dialog box, there are buttons for 'Exécuter' (Execute), 'Pas à pas détaillé' (Step-by-step detailed), 'Modifier' (Edit), 'Créer' (Create), 'Supprimer' (Delete), and 'Options...'. Below the list, it says 'Macros dans : Tous les classeurs ouverts' (Macros in: All open workbooks) and 'Description'. At the bottom right is an 'Annuler' (Cancel) button.

Commentaires

Il faut bien attaquer le langage VBA quelque part. Alors, nous allons partir sur des choses simples. Si vous connaissez déjà un autre langage de programmation, votre apprentissage en sera d'autant facilité. Dans le cas contraire, n'ayez crainte, vous vous en sortirez haut la main !

Commentaires

Les commentaires tout d'abord. Peut-être avez-vous remarqué les lignes en vert dans la macro **DateLongue** :

The screenshot shows the Microsoft VBA editor with the 'DateLongue' module selected. The code is as follows:

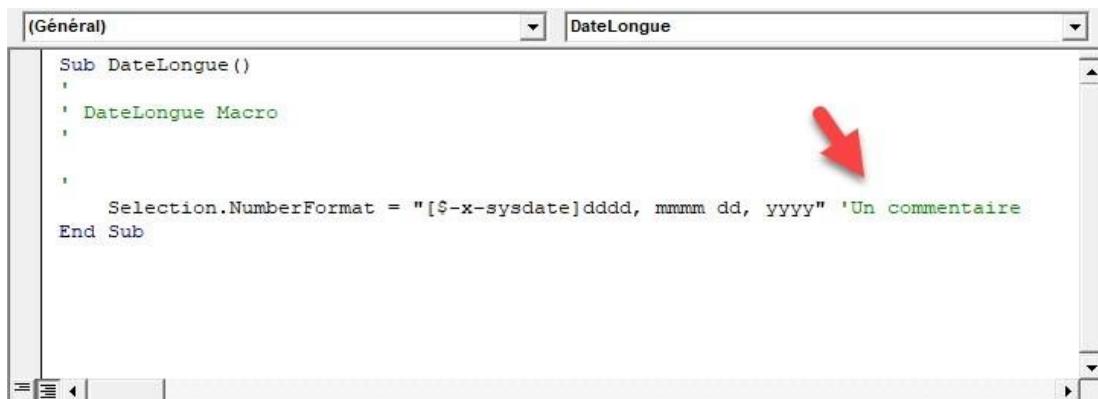
```

(Général) DateLongue
Sub DateLongue()
    ' DateLongue Macro
    '
    Selection.NumberFormat = "[\$-x-sysdate]dddd, mmmm dd, yyyy"
End Sub

```

A pink curly brace is placed after the first line of code, spanning the entire block of code, with the word 'Commentaires' written next to it in pink, indicating that the code is annotated with comments.

Ces lignes commencent par une apostrophe. Il s'agit de commentaires. Notez que les commentaires ne commencent pas obligatoirement au début d'une ligne : vous pouvez les placer à la suite d'une instruction, comme ici :



```
(Général) DateLongue
Sub DateLongue()
'
' DateLongue Macro
'
'
    Selection.NumberFormat = "[\$-x-sysdate]dddd, mmmm dd, yyyy" 'Un commentaire
End Sub
```

Si vous voulez définir un commentaire sur plusieurs lignes, vous devrez faire commencer chaque ligne par une apostrophe.

Il est parfois nécessaire de commenter plusieurs lignes qui contiennent des instructions VBA. Sélectionnez ces lignes, puis cliquez sur l'icône Commenter bloc dans la barre d'outils Edition :



Inversement, pour décommenter une ou plusieurs lignes, sélectionnez-les puis cliquez sur l'icône **Ne pas commenter bloc** dans la barre d'outils Edition :



Remarque

Si la barre d'outils **Edition** n'est pas affichée, déroulez le menu **Affichage**, pointez **Barres d'outils** et cliquez sur **Edition**.

Variabes

Les variables sont utilisées pour mémoriser des données. Comme leur nom l'indique, leur contenu pourra varier tout au long du code.

Vous pouvez déclarer vos variables au début d'un module, d'une procédure ou d'une fonction. Dans le premier cas, les variables déclarées pourront être utilisées dans toutes les

procédures et fonctions du module. Dans les deuxièmes et troisièmes cas, leur portée sera limitée à la procédure ou la fonction dans laquelle vous les avez définies.

Vous pouvez utiliser plusieurs types de variables en VBA. Nous les avons résumées dans le tableau suivant :

Type de variable	Description	Plage de valeurs
Byte	Stocke des entiers positifs.	0 à 255
Boolean	Représente une valeur booléenne, vraie ou fausse.	True ou False
Integer	Utilisé pour les nombres entiers (positifs ou négatifs).	-32 768 à 32 767
Long	Utilisé pour les grands nombres entiers.	-2 147 483 648 à 2 147 483 647
Single	Utilisé pour les nombres à virgule flottante simples.	-3.402823E38 à 3.402823E38
Double	Utilisé pour les nombres à virgule flottante doubles (précision plus grande que Single).	-1.79769313486231E308 à 1.79769313486231E308
Currency	Utilisé pour les valeurs monétaires (précision de 4 décimales).	-922 337 203 685 477.5808 à 922 337 203 685 477.5807
Decimal	Permet de stocker des valeurs décimales avec une grande précision (surtout pour les calculs financiers).	±79 228 162 514 264 337 593 543 950 à 79 228 162 514 264 337 593 543 950
String	Utilisé pour stocker du texte (chaînes de caractères).	Jusqu'à 2 milliards de caractères
Variant	Type de variable pouvant contenir différents types de données (numériques, chaînes, dates, etc.).	Dépend du contenu (par exemple, peut contenir des chaînes, des nombres, etc.)
Object	Représente un objet, comme une feuille de calcul, un graphique, une plage de cellules, etc.	Référence vers un objet
Date	Utilisé pour stocker une date ou une heure.	Du 1er janvier 100 à 31 décembre 9999
Time	Utilisé pour stocker uniquement l'heure.	De 00:00:00 à 23:59:59
Nothing	Spécifie une variable d'objet qui n'a pas encore été initialisée.	Aucune valeur (valeur par défaut d'une variable objet)

Pour définir une variable, vous utiliserez l'instruction **Dim** :

```
Dim entier As Integer
```

```
Dim texte As String
```

```
Dim booleen As Boolean
```

Vous pouvez définir plusieurs variables sur une seule ligne en les séparant par des virgules :

```
Dim v1 As Integer, c1 As String, r As Double
```

Lorsqu'une variable a été définie, vous pouvez lui affecter une valeur :

```
entier = 5  
  
texte = "une chaîne"  
  
r = 3.1415926536
```

Variables Static

Les variables sont généralement déclarées avec le mot-clé **Dim**. Cependant, dans certains cas, il peut être utile de les déclarer avec le mot-clé **Static**.

Lorsqu'une variable est déclarée avec le mot-clé **Static** dans une procédure ou une fonction, sa valeur est conservée à la fin de la procédure ou la fonction. Lorsque la procédure ou la fonction est à nouveau exécutée, la variable Static reprend la valeur qui avait été mémorisée.

Voici un exemple de code pour mieux comprendre le fonctionnement des variables **Static** :

```
Sub Incremente()  
  
    Static i As Integer  
  
    i = i + 1  
  
    Cells(i, 2) = i  
  
End Sub
```

Ici, la variable **Static i** de type **Integer** est définie dans la procédure **Incremente()**.

Lors de la première exécution de la procédure :

- La variable **i** vaut **0**.
- La deuxième instruction incrémente la variable **i**. Elle vaut donc **1**.
- La troisième instruction affiche la valeur de **i** dans la cellule à l'intersection de la ligne **i** et de la deuxième colonne. La cellule **B1** contient donc la valeur **1**.

Lors de la deuxième exécution de la procédure :

- Comme la variable **i** est statique, elle conserve la valeur qu'elle lors de la première exécution : **1**.

- La deuxième instruction l'incrémente. Elle vaut donc **2**.
- La troisième instruction affiche la valeur de **i** (**2**) dans la cellule à l'intersection de la ligne **i** (**2**) et de la deuxième colonne. La valeur **2** apparaît donc dans la cellule **B2**.

Ainsi de suite. Ici, le code a été exécuté 6 fois. Voici le résultat :

	A	B	C	D
1		1		
2		2		
3		3		
4		4		
5		5		
6		6		
7				
8				

Cellules, plages, feuilles et classeurs

Dans Excel, il est courant de manipuler des cellules, des plages de cellules, des feuilles de calcul et des classeurs. Dans cette section, vous allez découvrir les instructions utilisables pour les manipuler. Il ne s'agit que d'une introduction : vous irez beaucoup plus loin dans les sections à venir.

Les instructions suivantes permettent de sélectionner des feuilles, cellules, plages nommées, lignes et colonnes :

```
Sheets("Feuil2").Activate 'Activation de la feuille Feuil2  
Range("A3").Select 'Sélection de la cellule A3  
Range("B2:F6").Select 'Sélection du bloc de cellules B2 à F6  
Range("B2,F6").Select 'Sélection des cellules B2 et F6  
Range("unePlage").Select 'Sélection de la plage nommée unePlage  
Cells(4,2).select 'Sélection de la cellule à l'intersection de la ligne 4  
et de la colonne 2  
Range("3:5").Select 'Sélection des lignes 3 à 5  
Rows("3:5").Select 'Identique à l'instruction précédente  
Range("C:L").Select 'Sélection des colonnes C à L  
Columns("C:L").Select 'Identique à l'instruction précédente
```

Tableaux

Un tableau est une variable qui peut contenir plusieurs valeurs. Les différentes valeurs mémorisées sont accédées par leur indice.

Pour déclarer un tableau, vous utiliserez le mot-clé **Dim**. Ici par exemple, nous définissons un tableau d'entiers **T** qui peut contenir 26 valeurs, accessibles par les indices 0 à 25 :

```
Dim T(25) As Integer
```

Remarque

Si vous avez du mal avec l'indice des tableaux qui commence à 0, vous pouvez déclarer l'instruction suivante au début du module pour que l'indice du premier élément de tous les tableaux définis dans le module soit toujours 1 :

```
Option Base 1
```

Pour affecter la valeur 5 à la première case du tableau, vous utiliserez cette instruction :

```
T(0) = 5
```

Pour afficher dans la cellule **C2** le contenu de la première case du tableau, vous utiliserez cette instruction :

```
Cells(2,3) = T(0)
```

La cellule C2 affichera donc la valeur 5.

Définition et remplissage d'un tableau

Vous pouvez définir le contenu des cases d'un tableau avec la fonction **Array()**. Pour pouvoir utiliser cette fonction, le tableau doit avoir au préalable été déclaré de type **Variant**.

Par exemple, pour définir le tableau **Prenom** et lui affecter les trois prénoms suivants : **Emmanuel, Edi et Josué**, vous utiliserez les instructions suivantes :

```
Dim Prenom As Variant
```

```
Prenom = Array("Emmanuel", "Edi", "Josué")
```

Redimensionnement d'un tableau

Il est toujours possible de redimensionner un tableau en utilisant le mot-clé **Redim**.

Supposons que nous voulions ajouter une case au tableau **Prenom** précédent. Ce tableau comporte 3 cases. L'instruction à utiliser sera donc la suivante :

```
Redim Prenom(4)
```

Par défaut, le redimensionnement d'un tableau entraîne la perte des données qui y étaient stockées. Pour éviter de perdre les trois prénoms du tableau **Prenom**, vous utiliserez le mot-clé **Preserve** :

```
Redim Preserve Prenom(3)
```

Vous pourrez alors ajouter un quatrième prénom au tableau :

```
Prenom(3) = "Michel"
```

Affichez les quatre prénoms pour vous assurer que le redimensionnement n'a rien écrasé :

```
Cells(1,1) = Prenom(0)
```

```
Cells(1,2) = Prenom(1)
```

```
Cells(1,3) = Prenom(2)
```

```
Cells(1,4) = Prenom(3)
```

Voici le résultat :

	A	B	C	D
1	Emmanuel	Edi	Josué	Michel
2				
3				
4				
5				
6				

Tableaux multidimensionnels

Pour en terminer avec les tableaux, sachez qu'il est possible de définir des tableaux multidimensionnels. Pour cela, il suffit de définir les indices maximaux de chaque dimension lors de la définition du tableau. Ici par exemple, nous définissons un tableau à 3 dimensions :

```
Dim T(2,3,5) As Integer
```

Ce tableau contient $2 \times 3 \times 5$ cases, soit 30 cases. Par exemple, pour affecter la valeur 12 à la première case du tableau, vous utiliserez cette instruction :

```
T(0,0,0) = 12
```

Cellules, plages, feuilles et classeurs

Dans Excel, il est courant de manipuler des cellules, des plages de cellules, des feuilles de calcul et des classeurs. Dans cette section, vous allez découvrir les instructions utilisables pour les manipuler. Il ne s'agit que d'une introduction : vous irez beaucoup plus loin dans les sections à venir.

Les instructions suivantes permettent de sélectionner des feuilles, cellules, plages nommées, lignes et colonnes :

```
Sheets("Feuil2").Activate 'Activation de la feuille Feuil2  
  
Range("A3").Select 'Sélection de la cellule A3  
  
Range("B2:F6").Select 'Sélection du bloc de cellules B2 à F6  
  
Range("B2,F6").Select 'Sélection des cellules B2 et F6  
  
Range("unePlage").Select 'Sélection de la pahe nommée unePlage  
  
Cells(4,2).select 'Sélection de la cellule à l'intersection de la ligne 4 et de la colonne 2  
  
Range("3:5").Select 'Sélection des lignes 3 à 5  
  
Rows("3:5").Select 'Identique à l'instruction précédente  
  
Range("C:L").Select 'Sélection des colonnes C à L  
  
Columns("C:L").Select 'Identique à l'instruction précédente
```

Le modèle objet d'Excel

Si vous avez déjà côtoyé un langage de programmation objet, vous aurez certainement compris en lisant l'article précédent que les éléments manipulés dans Excel sont des objets.

Les principaux objets Excel

Si vous n'avez jamais approché de près ou de loin un langage objet, vous n'avez certainement aucune idée de ce qu'est un langage objet, ni comment le fait que le VBA soit un langage objet va impacter votre programmation. Eh bien, disons qu'Excel consiste en un ensemble de briques que nous appellerons « objets ». Par exemple, les classeurs, les feuilles de calcul, les plages et les cellules sont des objets Excel. Allons un peu plus loin :

- L'application Excel est un objet **Application**.
- Le classeur en cours est un objet **Workbook**.
- Les feuilles de calcul du classeur en cours sont des objets **Worksheet**.
- Une plage de cellule dans une feuille de calcul est un objet **Range**.
- Une cellule est un objet **Cell**.

Prenez une grande inspiration !

Les objets Excel appartiennent souvent à des « collections ». Par exemple :

Dans l'application Excel (objet **Application**), un ou plusieurs classeurs peuvent être ouverts. Ces classeurs constituent la collection **Workbooks**, c'est-à-dire l'ensemble des classeurs (objets **Workbook**) ouverts.

Un classeur (un **Workbook**) peut contenir une ou plusieurs feuilles de calcul. Ces feuilles de calcul constituent la collection **Worksheets**, c'est-à-dire l'ensemble des feuilles de calcul (les objets **Worksheet**) du classeur.

Une feuille de calcul contient un ensemble de cellules accessibles via des objets **Range**.

Les objets d'Excel – Quelques exemples pour bien comprendre

Un peu de pratique pour bien comprendre ce qui se passe.

Supposons que deux classeurs soient ouverts dans Excel. Le premier classeur s'appelle **classeur1.xlsx**. Il contient trois feuilles de calcul nommées **feuille1**, **feuille2** et **feuille3**. Le deuxième classeur s'appelle **classeur2.xlsx**. Il contient une feuille de calcul nommée **principal**.

Pour accéder à la feuille **feuille2** du classeur **classeur1.xlsx**, vous pouvez utiliser cette syntaxe :

```
Application.Workbooks("classeur1.xlsx").Worksheets("feuille2")
```

Ou plus simplement (l'objet Application étant implicite) :

```
Workbooks("classeur1.xlsx").Worksheets("feuille2")
```

Pour faire référence à un objet dans une collection, vous pouvez utiliser son nom, comme dans l'exemple précédent, mais également sa position dans la collection.

Par exemple, pour accéder à la feuille **feuille2** (qui occupe la deuxième position dans la collection **Worksheets**) du classeur **classeur1.xlsx**, vous pouvez utiliser cette syntaxe :

```
Application.Workbooks("classeur1.xlsx").Worksheets(2) Ou
```

plus simplement :

```
Workbooks("classeur1.xlsx").Worksheets(2) Voyons si vous avez compris.
```

Quelle instruction permet d'accéder à la cellule **C8** dans la première feuille de calcul de nom **Feuil1** du classeur **Classeur1.xlsx** ?

Deux syntaxes sont possibles :

```
Workbooks("classeur1.xlsx").Worksheets("Feuil1").Range("C8")
```

```
Workbooks("classeur1.xlsx").Worksheets(1).Range("C8")
```

Les propriétés et les méthodes des objets d'Excel

Pour vous représenter physiquement un objet Excel, imaginez que vous avez devant vous une boîte noire dont vous ne pouvez pas voir l'intérieur. Par contre, vous avez une documentation qui décrit les caractéristiques (on parle de propriétés) de l'objet. A l'aide de cette documentation, vous avez donc une vue globale de l'objet, sans pour autant en connaître les mécanismes. Mais après tout, c'est bien ce que vous voulez : utiliser Excel et pas comprendre comment il a été programmé !

Certaines propriétés sont accessibles en lecture seulement. D'autres en lecture et en écriture.

Vous utiliserez les premières pour connaître la valeur de telle ou telle propriété. Vous utiliserez les secondes pour connaître, mais aussi pour modifier les propriétés de l'objet.

Pour sélectionner l'objet ou les objets sur lesquels vous voulez lire ou modifier une propriété, vous utiliserez des **méthodes**. Par exemple, la méthode **Workbooks**(« **classeur1.xlsx** ») sélectionne le classeur **classeur1.xlsx**. Ou encore, la méthode **Range(B3-B5)** sélectionne la plage de cellules **B3 à B5**.

Pour utiliser une méthode, vous devez la plupart du temps faire référence à son parent. Par exemple, vous ne pouvez pas utiliser la méthode **Range()** sans préciser la feuille de calcul concernée. De même, vous ne pouvez pas utiliser la méthode **Worksheets()** sans préciser le classeur concerné. Par contre, lorsque vous sélectionnez un classeur, il n'est pas nécessaire d'indiquer son parent **Application** : cette liaison est implicite. La liaison parent-enfant se fait à l'aide d'une simple énumération, de la gauche vers la droite, de l'ancêtre le plus éloigné jusqu'à l'objet que vous voulez utiliser. L'expression se termine généralement par une propriété à laquelle vous affectez une valeur ou dont vous voulez connaître la valeur.

Par exemple, dans cette expression :

```
Workbooks("classeur1.xlsm").Worksheets("Feuil1").Range("C8") = "J'ai tout compris"
```

- `Workbooks("classeur1.xlsm")` sélectionne le classeur `classeur1.xlsm` ;
- `Worksheets("Feuil1")` sélectionne la feuille `Feuil1` dans le classeur `classeur1.xlsm` ;
- `Range("C8")` sélectionne la cellule `C8` dans la feuille `Feuil1` du classeur `classeur1.xlsm` ;
- `= "J'ai tout compris"` affecte une valeur texte à la cellule `C8` de la feuille `Feuil1` du classeur `classeur1.xlsm`.

Vous voyez, il n'y a rien de bien compliqué.

Le tout est de connaître :

- la hiérarchie des objets pour pouvoir les interconnecter dans une expression « à points » ;
- les méthodes et les propriétés des différents objets.

Les sections à venir vont progressivement vous familiariser avec les objets et les propriétés utilisables en VBA Excel, jusqu'à ce que vous vous sentiez « chez vous ». L'objectif sera alors atteint et il ne vous restera plus qu'à manipuler toutes ces entités pour faire des merveilles dans votre tableau !

La propriété Value d'un objet Range

Pour bien comprendre les mécanismes de base du VBA, raisonnons sur un exemple pratique.

La propriété **Value** d'un objet **Range** permet d'accéder en lecture et en écriture aux cellules concernées. Par exemple, l'instruction suivante affecte la valeur numérique 35 à la cellule C8 de la première feuille de calcul du classeur `Classeur1.xlsm` :

```
Workbooks("Classeur1.xlsm").Worksheets(1).Range("C8").Value = 35
```

La méthode `Range()` permet d'accéder à une cellule ou à une plage de cellules. Par exemple, pour affecter la valeur texte « bonjour » à la plage de cellules A1-C3, vous utiliserez cette instruction :

```
Workbooks("Classeur1.xlsm").Worksheets(1).Range("A1 :C3").Value = "bonjour"
```

Vous voulez savoir ce qui se trouve dans la cellule A2 ? Utilisez cette instruction :

```
Dim texte As String  
texte =  
Workbooks("Classeur1.xlsx") .Worksheets(1) .Range ("A2") .Value
```

Vous pourrez par la suite affecter à une autre cellule la valeur lue dans la cellule A2. Par exemple, à la cellule A3 :

```
Workbooks("Classeur1.xlsx") .Worksheets(1) .Range ("A3") .Value = texte
```

Que peut-on sélectionner dans une feuille de calcul ?

Lorsque vous utilisez Excel, vous sélectionnez fréquemment des cellules et des plages de cellules. Eh bien, VBA est en mesure d'effectuer les mêmes sélections.

Sélection d'une cellule

Deux fonctions peuvent être utilisées : **Cells()** et **Range()**. Leur syntaxe est très différente. A vous de décider quelle est celle qui vous convient le mieux.

Par exemple, cette instruction sélectionne la cellule à l'intersection de la ligne **4** et de la colonne **1** :

```
Cells(4,1) .Select
```

Ou encore, cette instruction sélectionne la cellule A4, qui se trouve à l'intersection de la ligne **4** et de la colonne **1** :

```
Range ("A4") .Select
```

Sélection d'une plage de cellules

La fonction Range peut également être utilisée pour sélectionner une plage de cellules. Par exemple, cette instruction sélectionne les cellules **A4** à **G12** :

```
Range ("A4:G12") .Select
```

Sélection de plusieurs plages de cellules

La fonction **Range()** permet également de sélectionner plusieurs plages de cellules. Par exemple, pour sélectionner les cellules **A4** à **B5** et les cellules **B9** à **D11**, vous utiliserez cette instruction :

```
Range ("A4:B5,B9:D11") .Select
```

Sélection d'une plage de cellules nommées

Lorsqu'une plage de cellules est nommée, vous pouvez la sélectionner en précisant son nom dans la fonction **Range()**. Supposons que la plage de cellules A8:E8 ait pour nom **resultats**. Pour la sélectionner en VBA, vous utiliserez cette instruction :

```
Range("resultats").Select
```

Pour que cette instruction fonctionne, la plage **resultats** doit avoir été définie, sans quoi, une erreur se produira à l'exécution :



Le nom de la plage n'est pas sensible à la casse des caractères. Les instructions suivantes sont donc tout aussi valables pour sélectionner la plage **A8:E8** :

```
Range("Resultats").Select
```

```
Range("RESULTATS").Select
```

Sélection de lignes et de colonnes

Les fonctions **Rows()** et **Columns()** permettent de sélectionner une ou plusieurs lignes et colonnes. Par exemple, pour sélectionner la colonne C, vous écrirez :

```
Columns("C").Select
```

Pour sélectionner les colonnes C à G, vous écrirez :

```
Columns("C:G").Select
```

D'une façon similaire, pour sélectionner la ligne 3, vous écrirez :

```
Rows("3").Select
```

Ou encore, pour sélectionner les lignes 3 à 7, vous écrirez :

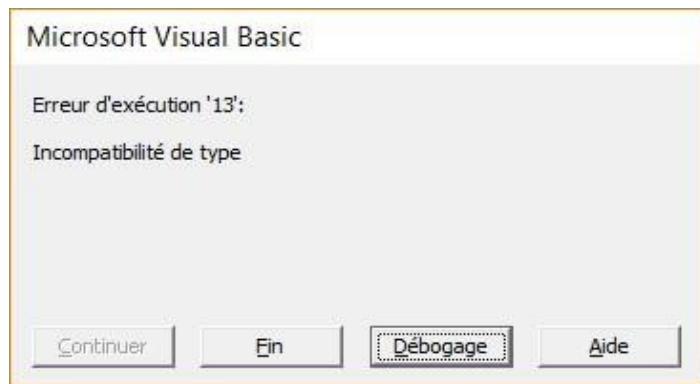
```
Rows("3:7").Select
```

Sélection de lignes et de colonnes disjointes

Supposons que vous vouliez sélectionner les lignes 4, 9 et 10. Vous pouvez tenter quelque chose comme ceci :

```
Rows ("4,9,10").Select
```

Pas de chance, cette instruction produit une erreur à l'exécution :



Vous utiliserez plutôt la fonction **Range()** :

```
Range ("4:4,9:10").Select
```

De même, supposons que vous vouliez sélectionner les colonnes **B, D et F à H**, vous utiliserez cette instruction :

```
Range ("B:B,D:D,F:H").Select
```

Sélection de toute la feuille de calcul

Pour sélectionner toutes les cellules de la feuille de calcul, vous utiliserez cette instruction :

```
Cells.Select
```

xlsx, xlsm, xlsb ?

Depuis la sortie d'Excel 2007, les classeurs peuvent être stockés avec une extension xlsx, xlsm ou xlsb.

- Les fichiers **xlsx** ne peuvent pas contenir du code VBA.
- Les fichiers **xlsm** peuvent contenir du code VBA.
- Les fichiers **xlsb** (classeurs Excel binaire) peuvent contenir ou ne pas contenir du code VBA. Contrairement aux deux autres formats de fichiers, ils sont compressés et prennent donc moins de place sur le disque.

En ce qui concerne cette série de sections, vous sauvegarderez vos classeurs au format **xlsm**, sans quoi, le code VBA ne sera pas mémorisé.

Travailler avec des sélections

La méthode **Range()** cible une cellule ou une plage de cellules. Dans un article précédent, vous avez appris à modifier le contenu de cette ou de ces cellules *via* la propriété **Value**.

Si vous voulez ne modifier qu'une des cellules sélectionnées, vous devez au préalable l'activer avec la méthode **Activate** et lui affecter une valeur avec **ActiveCell.Value**. Par exemple, ces instructions sélectionnent la plage **A2:C5**, activent la cellule **B4**, affectent la valeur **10** aux cellules sélectionnées et la valeur **20** à la cellule active :

```
Range ("A2:C5") .Select
```

```
Range ("B4") .Activate
```

```
Selection.Value = 10
```

```
ActiveCell.Value = 20
```

Voici le résultatat de ces instructions :

	A	B	C	D
1				
2	10	10	10	
3	10	10	10	
4	10	20	10	
5	10	10	10	
6				
7				

Sur la plage de cellules sélectionnée ou sur la cellule active, vous pouvez modifier (entre autres) :

- La couleur d'écriture avec **Color**
- La couleur d'arrière-plan avec **Color**
- L'alignement horizontal avec **HorizontalAlignment**
- La police de caractères avec **Name**
- Les attributs gras et italique en affectant la valeur **True** aux propriétés **Bold** et **Font.Italic**

Voici quelques exemples de code :

```
Range ("A2:C5") .Select 'Sélection de la plage A2:C5  
Selection.Interior.Color = RGB(255, 0, 0) ' Arrière-plan rouge sur la sélection
```

```
Selection.HorizontalAlignment = xlCenter 'Centrage horizontal de la sélection
```

```
Selection.Font.Name = "Courier" ' Police Courier sur la sélection  
Selection.Font.Bold = True 'Sélection en gras  
Selection.Font.Italic = True 'Sélection en italique
```

Toutes ces instructions fonctionneraient également sur la cellule active :

```
Range("B5").Activate 'Sélection de la cellule B5  
  
ActiveCell.Interior.Color = RGB(255, 0, 0) 'Arrière-plan de la cellule active  
  
ActiveCell.HorizontalAlignment = xlLeft 'Alignement de la cellule active  
ActiveCell.Font.Name = "Algerian" ' Police de la cellule active  
  
ActiveCell.Font.Bold = True ' Cellule active en gras  
  
ActiveCell.Font.Italic = True ' Cellule active en italique
```

Pour éviter d'avoir à répéter Selection ou ActiveCell, vous pourriez factoriser ces deux objets.

Ce qui donnerait avec **Selection** :

```
Range("A2:C5").Select  
  
With Selection  
  
.Interior.Color = RGB(255, 0, 0)  
  
.HorizontalAlignment = xlCenter  
  
.Font.Name = "Courier"  
  
.Font.Bold = True  
  
.Font.Italic = True
```

End With

Et avec **ActiveCell** :

```
Range("B5").Activate  
  
With ActiveCell  
  
.Interior.Color = RGB(0, 255, 0)  
  
.HorizontalAlignment = xlLeft  
  
.Font.Name = "Algerian"
```

```

    .Font.Bold = True

    .Font.Italic = True

End With

```

Travailler avec des couleurs

En VBA Excel, les couleurs peuvent être choisies dans un panel de 56 couleurs prédéfinies ou créées en mélangeant une certaine quantité des trois couleurs primaires : rouge, vert et bleu.

Utilisation des couleurs prédéfinies

Pour affecter une de des couleurs au texte ou à l'arrière-plan de la sélection ou de la cellule active, vous utiliserez ces expressions :

```

Range("A10").Select 'Sélection de la cellule A10

Selection.Interior.ColorIndex = 4 'Arrière-plan de couleur 4

Selection.Font.ColorIndex = 26 'Police de couleur 26
Range("A12").Activate 'Activation de la cellule A12
ActiveCell.Interior.ColorIndex = 4 'Arrière-plan de couleur 4
ActiveCell.Font.ColorIndex = 26 'Police de couleur 26

```

Utilisation des couleurs RGB

La couleur du texte ou de l'arrière-plan peut être définie avec la fonction **RGB()** :

`RGB(rouge, vert, bleu)`

Où **rouge**, **vert** et **bleu** sont la quantité de rouge de vert et de bleu qui constituent la couleur. Ces trois valeurs doivent être comprises entre **0** et **255**. La valeur **0** signifie que la couleur est absente. La valeur **255** signifie que la couleur est maximale.

Etant donné que chacune de ces composantes peut prendre 256 valeurs, la fonction **RGB()** donne accès à $256 \times 256 \times 256$ couleurs, soit 16 777 216 couleurs !

Voici quelques exemples :

Valeur	Couleur
<code>RGB(0,0,0)</code>	Noir
<code>RGB(255,255,255)</code>	Blanc
<code>RGB(255,0,0)</code>	Rouge
<code>RGB(0,255,0)</code>	Vert
<code>RGB(0,0,255)</code>	Bleu

RGB(255,255,0)	Jaune
----------------	-------

Comment ferez-vous pour définir un gris ?

Pour cela, vous devez prendre la même quantité pour chaque couleur primaire.

Vous pourrez ainsi définir 256 niveaux de gris, depuis le gris très foncé (noir) défini par **RGB(0,0,0)** au gris très clair (blanc) défini par **RGB(255,255,255)**. Vous définirez par exemple un gris foncé avec **RGB(50,50,50)** et un gris clair avec **RGB(200,200,200)**.

Pour affecter une couleur de texte ou d'arrière-plan à la sélection ou à la cellule active, vous utiliserez des instructions de ce type :

```
Range("A10").Select 'Sélection de la cellule A10
Selection.Interior.Color = RGB(255,0,0) 'Arrière-plan de couleur rouge
Selection.Font.Color = RGB(0,255,0) 'Police de couleur verte
Range("A12").Activate 'Activation de la cellule A12
ActiveCell.Interior.Color = RGB(255,0,0) 'Arrière-plan de couleur rouge
ActiveCell.Font.Color = RGB(0,255,0) 'Police de couleur verte
```

Tests

Cette section va vous montrer comment effectuer des tests en VBA.

If Then Else

Il est parfois nécessaire d'exécuter une ou plusieurs instructions lorsqu'une condition est vérifiée, et éventuellement une ou plusieurs autres instructions dans le cas contraire. Pour cela, vous utiliserez une instruction **If Then Else**.

La syntaxe de l'instruction If Then Else

Voici la syntaxe de l'instruction **If Then Else**:

```
If condition Then
    ' Une ou plusieurs instructions
Else
    ' Une ou plusieurs instructions
End If
```

Le bloc **Else** ne sera pas nécessaire si une ou plusieurs instructions ne doivent pas être exécutées lorsque la condition n'est pas vérifiée. L'instruction se simplifie :

```
If condition Then  
    ' Une ou plusieurs instructions  
End If
```

Voici un exemple de code :

```
Dim entier As Integer  
entier = 6  
  
If entier = 5 Then  
  
    MsgBox "entier vaut 5"  
  
Else  
  
    MsgBox "entier est différent de 5"  
  
End If
```

Ici, on définit la variable Integer **entier** et on lui affecte la valeur **6**. Par la suite, on teste si sa valeur est égale à **5** avec une instruction **If Then Else**. Comme ce n'est pas le cas, une boîte de message s'affiche et indique « entier est différent de 5 ».

Les opérateurs de comparaison utilisables dans un test

L'opérateur « = » n'est pas le seul utilisable dans un test **If Then Else**. Le tableau dresse la liste des opérateurs utilisables.

Opérateur	Signification	Exemple
=	Test d'égalité	If a = 12
<	Test inférieur à	If a < 12
<=	Test inférieur ou égal à	If a <= 12
>	Test supérieur à	If a > 12
>=	Test supérieur ou égal à	If a >= 12
<>	Test différent de	If a <> 12

Ces opérateurs peuvent être appliqués sur :

- des nombres (Byte, Integer, Long, Currency, Single ou Double) ;
- des chaînes (String) ;
- des dates (Date).

Voici quelques exemples :

```
Dim n As Integer, s As String, d As Date
```

```

n = 1.2545

s = "Un peu de texte"

d = "12/08/2110"

If d > "10/05/2020" Then

    MsgBox "pas tout de suite..."

End If

If n = 1.2545 Then

    MsgBox "n est bien égal à 1.2545"

End If

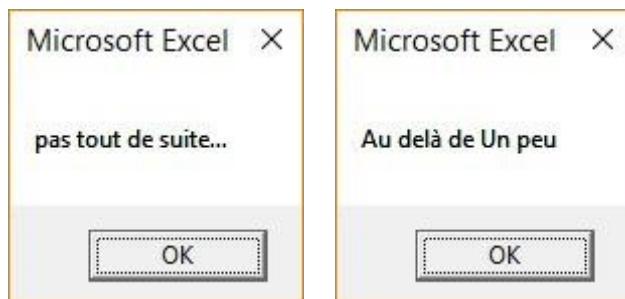
If s > "Un peu" Then

    MsgBox "Au delà de Un peu"

End If

```

Si vous exécutez ce code, vous serez peut-être surpris, car il affiche les boîtes de dialogue suivantes :



La première boîte de dialogue est facile à comprendre. En effet, la date **12/08/2110** est postérieure à **10/05/2020**. Le test **If d > « 10/05/2020 » Then** est donc vérifié et le message s'affiche.

Par contre, que devrait donner le test **If n = 1.2545 Then** selon vous ? Etant donné que **n** a été initialisé à **1.2545** un peu plus haut, le message « **n** est bien égal à **1.2545** devrait s'afficher. Eh bien non !

Pourquoi d'après vous ?

Tout simplement parce que la variable **n** a été définie en tant qu'**Integer**. Lors de son affectation, la valeur décimale a purement et simplement été supprimée.

Remplacez **Integer** par **Single** dans cette expression et le test devrait aboutir :

```
Dim n As Integer, s As String, d As Date
```

Que pensez-vous du troisième test ? Comment deux chaînes pourraient être comparées avec l'opérateur « > » ? Est-ce que « **Un peu de texte** » > « **Un peu** » ?

Selon VBA, oui !

La comparaison de deux chaînes se fait selon un ordre alphabétique. Ainsi :

A < B < E < Z

Mais aussi :

A < Abc < Bcd < Be < Htzert < Huv

Notez que l'instruction **Option Compare** peut changer les choses lorsque vous comparez des chaînes.

Après cette instruction :

```
Option Compare Binary
```

Les caractères sont comparés par rapport à leur représentation binaire. Ainsi :

A < B < E < Z < a < b < e < z < À < Â < Ø < à < ê < ø

Après cette instruction :

```
Option Compare Text
```

Les caractères sont comparés sans distinctions entre les majuscules et les minuscules :

(A=a) < (À=à) < (B=b) < (E=e) < (Ê=ê) < (Ø = ø) < (Z=z)

Comparaison de chaînes avec l'opérateur Like

L'opérateur **Like** est très puissant. Il permet de comparer une variable **String** et un modèle. La syntaxe générale de cette instruction est la suivante :

```
b = s Like modèle
```

Où **b** est une variable **Boolean**, **s** est une variable **String** et **modèle** est le modèle à appliquer à la chaîne **s**. Si la variable **s** correspond au modèle, la variable **b** est initialisée à **True**. Elle est initialisée à **False** dans le cas contraire.

Le modèle peut contenir un ou plusieurs caractères quelconques (des lettres, des chiffres ou des signes), mais aussi des caractères de remplacement :

Caractère de remplacement	Signification
---------------------------	---------------

?	Un caractère quelconque
*	zéro, un ou plusieurs caractères quelconques
#	Un chiffre quelconque
[A-Z]	Une lettre majuscule quelconque
[A-CT-Z]	Une lettre majuscule comprise entre A et C ou entre T et Z
[A-Za-z0-9]	Une lettre majuscule ou minuscule quelconque ou un chiffre
[!G-J]	Une lettre majuscule différente de G, H, I et J

Par exemple, pour tester si la cellule **B5** contient un **nombre composé de 5 chiffres**, vous utiliserez le modèle « ##### ». Voici ce que vous pourriez écrire :

```
Dim n As Integer
n = Range("B5")
MsgBox n
If n Like "#####" Then
    MsgBox "Vous avez bien entré un nombre à 5 chiffres"
Else
    MsgBox "Vous n'avez pas entré un nombre à 5 chiffres"
End If
```

Opérateurs logiques

Pour compléter vos tests, vous pouvez également utiliser des opérateurs logiques :

Opérateur logique	Signification	Exemple
And	Et logique	If a=5 And b<12 Then
Or	Ou logique	If a=5 Or b<12 Then
Not	Non logique	If Not (a = 6) Then
Is	Comparaison de deux objets VBA	If Worksheets(1) Is Worksheets(2)

Test multiple Select Case

Supposons qu'une variable **v** puisse prendre plusieurs valeurs et que vous deviez effectuer un traitement spécifique pour chaque valeur. Dans ce cas, l'instruction **If Then Else** n'est pas appropriée. Vous utiliserez plutôt cette instruction :

```
If v = valeur1 Then
```

```

' Traitement 1

ElseIf v = valeur2 Then

' Traitement 2

ElseIf v = valeur3 Then

' Traitement 3

ElseIf v = valeur4 Then

' Traitement 4

ElseIf v = valeur5 Then

' Traitement 5

Else

' Traitement 6

End If

```

Si vous devez tester trois ou plus de trois valeurs, je vous conseille d'utiliser une instruction **Select Case** à la place du **If Then ElseIf**. Le test précédent devient donc :

```

Select Case v

Case valeur1

' Traitement1

Case valeur2

' Traitement2

Case valeur3

' Traitement3

Case valeur4

' Traitement4

Case valeur5

' Traitement5

Case Else

```

```
' Traitement6  
End Select
```

Voici quelques exemples de l'instruction **Case** :

```
Case 5 'La valeur numérique 5  
Case 10 To 20 'Entre 10 et 20  
Case Is > 20 'Supérieur à 20  
Case "a" 'La chaîne "a"  
Case "a", "b", "c" ' La chaîne "a", "b" ou "c" Attention
```

Avec les opérateurs `<`, `>` et `<>`, vous devez utiliser un **Case Is** et non un **Case**.

Les traitements peuvent être des instructions VBA quelconques. Par exemple :

```
a = 5 ' Affectation de la valeur 5 à la variable a  
MsgBox "texte" ' Affichage d'une boîte de message  
Sheets("Feuil2").Activate 'Activation de la feuille Feuil2
```

Vous pouvez également utiliser une instruction pour faire une pause dans l'exécution ou pour l'arrêter :

```
Stop ' Arrêt sur cette instruction  
End ' Fin du code
```

Boucles

En programmation, il est souvent nécessaire d'exécuter une portion de code plusieurs fois de suite. La répétition des instructions peut se produire un certain nombre de fois connu à l'avance ou tant qu'une condition est vérifiée. Cette section va passer en revue toutes les instructions de bouclage du langage VBA.

La boucle For ... Next

Lorsque le nombre de répétitions est connu à l'avance, le plus simple est d'utiliser une boucle **For ... Next**. Voici sa syntaxe :

```
For compteur = début To Fin Step pas
```

' Une ou plusieurs instructions

Next compteur

Où **Compteur** est une variable qui commence à **début** et finit à **fin** en progressant de **pas** à chaque itération de la boucle.

Remarque

Si la partie **Step pas** n'est pas précisée, la variable compteur va de **début** à **fin** par pas de **1**.

Quelques exemples pour mieux comprendre.

Supposons que vous vouliez remplir les cellules **A1** à **H1** avec les valeurs **1** à **8**. Voici le code à utiliser :

```
Dim i As Integer
```

```
For i = 1 to 8
```

```
    Cells(1, i) = i
```

```
Next i
```

Ici, nous utilisons la méthode **Cells()** en précisant le numéro de ligne et le numéro de colonne. Voici le résultat :

	A	B	C	D	E	F	G	H	I
1	1	2	3	4	5	6	7	8	
2									
3									

Supposons maintenant que vous vouliez remplir les cellules **C3** à **C12** avec les valeurs « **a** » à « **j** ». Voici le code à utiliser :

```
Dim i As Integer
```

```
For i = 3 To 12
```

```
    Cells(i, 3) = Chr(97 + i - 3)
```

```
Next i
```

Ici, nous utilisons la méthode **Cells()** en précisant le numéro de ligne (**i**, qui va de **3** à **12**) et le numéro de colonne (fixe et égal à **3**). La méthode **Chr()** convertit le code ASCII passé en argument en un caractère. Pour arriver au résultat souhaité, le code ASCII de la lettre « **a** » a été cherché dans un tableau de codes ASCII. Par exemple sur la page

<http://www.tableascii.com/> :

091	133	5B	01011011	[(lef^
092	134	5C	01011100	\	(bac
093	135	5D	01011101]	(rig
094	136	5E	01011110	^	(car
095	137	5F	01011111	-	(und
096	140	60	01100000		
097	141	61	01100001	a	
098	142	62	01100010	b	
099	143	63	01100011	c	
100	144	64	01100100	d	
101	145	65	01100101	e	
102	146	66	01100110	f	
103	147	67	01100111	g	
104	150	68	01101000	h	
105	151	69	01101001	i	
106	152	6A	01101010	j	
107	153	6B	01101011	k	
108	154	6C	01101100	l	
109	155	6D	01101101	m	
110	156	6F	01101110	n	

Etant donné que **i** varie de **3** à **12**, la formule à utiliser est **97 + i – 3**. L'argument de la fonction **Chr()** va donc de **97** (pour **i = 3**) à **106** (pour **i = 12**).

Voici le résultat :

	A	B	C	D
1				
2				
3		a		
4		b		
5		c		
6		d		
7		e		
8		f		
9		g		
10		h		
11		i		
12		j		
13				
14				

Supposons maintenant que vous voulez remplir le bloc de cellules **A3-C9** comme ceci :

	A	B	C	D
1				
2				
3	1	2	3	
4	4	5	6	
5	7	8	9	
6	10	11	12	
7	13	14	15	
8	16	17	18	
9	19	20	21	
10				
11				
12				

Comment feriez-vous ?

Observez la progression numérique dans les cellules. Les valeurs vont de **1** à **21**, et on change de ligne toutes les trois cellules.

Pour faire simple, nous allons imbriquer deux boucles : une pour les lignes et une pour les colonnes. Voici le code :

```
Dim i, j As Integer

For i = 0 To 6
    For j = 1 To 3
        Cells(i + 3, j) = j + i * 3
    Next j
Next i
```

La boucle la plus extérieure s'intéresse aux lignes et la boucle la plus intérieure aux colonnes. Les index des boucles ont été choisis pour simplifier au maximum la formule.

Lors de la première exécution de la boucle extérieure, **i** vaut **0** et **j** va de **1** à **3**. La formule :

`Cells(i + 3, j) = j + i * 3`

Cible donc les cellules **A3**, **B3** puis **C3**. Et la valeur affectée à ces cellules est **1**, **2** et **3**. Si vous ne voyez pas ce que je veux dire, simulez le fonctionnement de la boucle interne en remplaçant **j** par **1**, **2** puis **3** et voyez le résultat. Vous obtenez :

- Pour **i=0** et **j=1** : **Cells(3,1) = 1**, soit **A3 = 1**
- Pour **i=0** et **j=2** : **Cells(3,2) = 2**, soit **B3 = 2**
- Pour **i=0** et **j=3** : **Cells(3,3) = 3**, soit **C3 = 3**

Lorsque la boucle interne a fini de s'exécuter, la boucle externe incrémente la valeur de **i**, qui passe de **0** à **1**. La formule :

`Cells(i + 3, j) = j + i * 3`

Cible alors les cellules **A4**, **B4** et **C4** et leur affecte (respectivement) les valeurs **4**, **5** et **6**.

La progression de **i** et de **j** se poursuit jusqu'à ce que ces index dépassent les valeurs maximales fixées à **6** et **3**. Les deux boucles prennent alors fin et la plage **A3-C9** est entièrement remplie.

Un dernier exemple pour illustrer l'utilisation d'un pas de progression. Supposons que vous vouliez obtenir le résultat suivant :

	A	B	C	D	E	F	G
1	2	4	6	8	10	12	
2							
3							
4							

Plusieurs approches sont possibles, mais une des plus simples consiste certainement à utiliser un pas de progression négatif dans la boucle :

```
Dim i As Integer  
  
For i = 12 To 2 Step -2  
  
    Cells(1, i / 2) = i  
  
Next i
```

Ici, la variable **i** passe de **12** à **2** par pas de **-2**. Elle vaut donc successivement **12, 10, 8, 6, 4, puis 2**.

La formule :

```
Cells(1, i / 2) = i
```

Cible les cellules situées en ligne **1** et en colonne **6** ($12/2$), **5** ($10/2$), **4** ($8/2$), **3** ($6/2$), **2** ($4/2$) puis **1** ($2/2$). La valeur de **i** est affectée à ces cellules, ce qui donne bien le résultat attendu.

Dans certaines boucles, on ne sait pas à l'avance combien de fois les instructions seront exécutées. Dans ce cas, la boucle **For ... Next** n'est pas appropriée. Vous utiliserez plutôt une boucle **For Each... Next**, **While ... Wend**, **Do While ... Loop**, **Do ... Loop While** ou **Do Until ... Loop**.

La boucle For Each... Next

La boucle **For Each ... Next** permet d'itérer sur une collection d'objets, comme les cellules d'une plage, les feuilles d'un classeur ou les objets d'un tableau. Voici sa syntaxe :

```
For Each element In collection
```

' Une ou plusieurs instructions

Next element

Par exemple, pour remplir les cellules A1 à H1 avec les chiffres 1 à 8 en utilisant une boucle For Each ... Next, on peut écrire :

```
Dim cellule As Range  
  
Dim valeurs As Variant  
  
Dim i As Integer  
  
valeurs = Array(1, 2, 3, 4, 5, 6, 7, 8)  
  
i = 0  
  
For Each cellule In Range("A1:H1")  
    cellule.Value = valeurs(i)  
  
    i = i + 1  
  
Next cellule
```

La boucle While ... Wend

La boucle **While ... Wend** est exécutée tant qu'une condition est vérifiée. Voici sa syntaxe :

While condition

' Une ou plusieurs instructions

Wend

Par exemple, pour remplir les cellules **A1 à H1** avec les chiffres **1 à 8** avec une boucle **While ... Wend**, comme dans la copie d'écran suivante :

	A	B	C	D	E	F	G	H	I
1	1	2	3	4	5	6	7	8	
2									
3									

Vous pourriez utiliser ces instructions :

```
Dim i As Integer  
i = 1
```

```
While i < 9
```

```
    Cells(1, i) = i
```

```
i = i + 1
```

```
Wend
```

Ces instructions peuvent sembler plus complexes que celles utilisées dans la boucle **For ... Next** équivalente. Examinons-les pas à pas.

- La variable **i** est déclarée en tant qu'**Integer** et initialisée à **1**.
- La boucle se poursuit tant que **i** est inférieur à **9**.
- Les cellules de la ligne **1** et de la colonne **1 à 8** sont alors ciblées avec la méthode **Cells()** et on leur affecte la valeur de la variable **i**, c'est-à-dire **1 à 8**.
- La variable **i** est incrémentée d'un à chaque passage dans la boucle avec l'instruction **i = i + 1**.

Vous voyez, il n'y a rien de bien compliqué.

La boucle Do While ... Loop

Nous allons maintenant utiliser une boucle **Do While ... Loop** pour obtenir le même résultat que dans la boucle **While ... Wend** précédente. Voici le code utilisé :

```
Dim i As Integer  
i = 1
```

```
Do While i < 9
```

```
    Cells(1, i) = i
```

```
    i = i + 1
```

```
Loop
```

Le code est strictement équivalent. Il n'y a que l'écriture qui change.

La boucle Do ... Loop While

Voyons maintenant comment utiliser une boucle **Do ... Loop While** pour obtenir le même résultat. Voici le code utilisé :

```
Dim i As Integer  
i = 1  
Do
```

```
    Cells(1, i) = i
```

```
    i = i + 1
```

```
Loop While i < 9
```

Ici, la condition de fin de boucle est testée en fin de boucle. Quelle que soit la valeur de **i**, la boucle s'exécute donc au moins une fois. Mis à part cette légère différence, le code est très semblable à celui utilisé dans les boucles précédentes.

La boucle Do Until ... Loop

Voyons enfin comment utiliser une boucle **Do Until ... Loop** pour arriver au même résultat. Voici le code utilisé :

```
Dim i As Integer  
i = 1
```

```
Do Until i > 8
```

```
    Cells(1, i) = i
```

```
    i = i + 1
```

```
Loop
```

Ici, le test de fin de boucle est effectué en début de boucle. La boucle prend fin quand **i** est **supérieur à 8** (et non tant que **i** est inférieur à 9 comme dans les autres boucles). Mis à part ce détail, le code est similaire à celui utilisé dans les autres boucles.

Maintenant que vous connaissez les différentes boucles utilisables en VBA, vous devrez trouver celle qui est la plus appropriée à chaque cas que vous devrez traiter...

Quitter une boucle prématurément

Cette section ne serait pas complet si nous ne parlions pas des sorties prématurées des boucles. Rien de bien sorcier rassurez-vous.

Supposons que lorsqu'une condition est vérifiée, vous vouliez terminer la boucle. Dans ce cas, vous utiliserez une des instructions suivantes :

Boucle	Instruction pour sortir de la boucle
For ... Next	Exit For
For Each ... Next	Exit For
While ... Wend	Exit While
Do While ... Loop	Exit Do
Do ... Loop While	Exit Do
Do Until ... Loop	Exit Do

Gestion d'erreurs en VBA

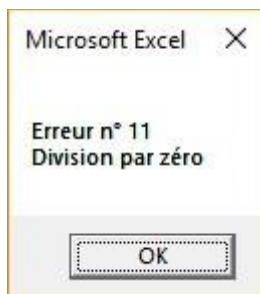
Des erreurs peuvent parfois apparaître lors de l'exécution d'un code VBA. Si vous ne mettez pas en place un gestionnaire d'erreur, une boîte de dialogue peu esthétique stoppera net le fonctionnement du programme.

Mise en place d'un gestionnaire d'erreurs

Pour définir un gestionnaire d'erreurs dans une procédure, vous utiliserez une instruction **On Error GoTo**. Voici un exemple de code :

```
Sub uneProcedure()  
  
    On Error GoTo gestionErreurs  
  
    Dim a As Integer  
  
    a = 1 / 0  
gestionErreurs:  
  
    MsgBox "Erreur n° " & Err.Number & vbCrLf & Err.Description  
  
End Sub
```

Et voici le résultat :



Ici, on tente d'affecter la valeur 1/0 à la variable Integer **a**. Ce qui produit une erreur 11 : Division par zéro.

Annulation du gestionnaire d'erreurs

A tout moment, vous pouvez désactiver le gestionnaire d'erreur mis en place par une instruction **On Error GoTo**. Pour cela, il suffit d'utiliser cette instruction :

```
On Error GoTo 0
```

Examinez ce code :

```
Sub uneProcedure()  
  
    On Error GoTo gestionErreurs  
  
    Dim a As Integer
```

```

a = 1 / 0
gestionErreurs:

MsgBox "Erreur n° " & Err.Number & vbCrLf & Err.Description

On Error GoTo 0

a = 2 / 0
End Sub

```

La première division par zéro provoque l'exécution du gestionnaire d'erreurs. Ce dernier :

- affiche un message d'erreur ;
- désactive le gestionnaire d'erreur ;
- lance intentionnellement une instruction qui provoque une erreur.

Voici le résultat :



Le deuxième message d'erreur est beaucoup moins engageant que le premier !

Reprise de l'exécution

Placée dans le gestionnaire d'erreurs, l'instruction **Resume** reprend l'exécution sur l'instruction qui a provoqué une erreur.

Examinez ce code :

```

Sub uneProcedure()

On Error GoTo gestionErreurs

Dim a As Integer

```

```

a = InputBox("Entrez un entier")

End
gestionErreurs:

MsgBox "Vous devez saisir un entier"

Resume

End Sub

```

Ici, la saisie de l'utilisateur est affectée à une variable de type **Integer**. Si l'utilisateur n'entre pas un nombre entier, une erreur se produira, et le gestionnaire d'erreurs sera exécuté.

Dans ce cas, un message d'erreur demandant à l'utilisateur de saisir un entier sera affiché et l'instruction **Resume** provoquera la réexécution de la ligne : `a = InputBox("Entrez un entier")`

Si l'utilisateur entre un nombre entier, l'instruction qui suit le **InputBox()** sera exécutée et la procédure se terminera.

Poursuite de l'exécution

Plutôt que d'indiquer un gestionnaire d'erreurs dans l'instruction **On Error GoTo**, vous pouvez demander d'ignorer l'erreur et d'exécuter la ligne suivante :

```
On Error Resume Next
```

Par exemple, pour tester si la feuille « F3 » existe dans le classeur courant, vous pourriez utiliser ce code :

```

Sub uneProcedure()

    Dim feuille As Worksheet

    On Error Resume Next

    Set feuille = Worksheets("F3")

    If feuille Is Nothing Then

        MsgBox "La feuille ""F3"" n'existe pas"
    Else

        MsgBox "La feuille ""F3"" existe"

    End If

End Sub

```

L'instruction

```
On Error Resume Next
```

fait en sorte que le code ne s'arrête pas sur une erreur d'exécution. L'erreur est purement et simplement ignorée et l'exécution se poursuit sur la ligne suivante.

L'instruction

```
Set feuille = Worksheets("F3")
```

tente d'affecter à la variable feuille la feuille **F3**. Si cette feuille n'existe pas, l'erreur est ignorée (grâce à l'instruction **On Error Resume Next**) et l'instruction suivante s'exécute.

Ici, on teste si la variable feuille fait ou ne fait pas référence à une feuille :

```
If feuille Is Nothing Then  
Else  
End If
```

Selon le résultat du test, un message ou un autre est affiché avec une instruction **MsgBox()**.

Messages

Lorsqu'on commence à écrire du code VBA, il est fréquent de vouloir afficher des données textuelles ou numériques dans une boîte de message. La méthode **MsgBox()** est là pour ça. Comme vous le verrez dans cette section, vous pouvez également utiliser **MsgBox()** en tant que fonction pour poser une question à l'utilisateur. En fonction de son choix, vous pourrez alors exécuter un bloc d'instructions ou un autre.

La méthode **MsgBox()**

Pour afficher une boîte de message, vous utiliserez **MsgBox()** en tant que méthode. Voici quelques exemples de code :

Affichage d'un message texte

```
MsgBox "Un message texte"
```



Affichage du contenu d'une cellule avec la fonction **Cells()**

```
MsgBox Cells(1, 2)
```

	A	B	C
1	1 A		
2	2 B		
3	3 C		
4			
5			

Affichage du contenu d'une cellule avec la fonction Range()

```
MsgBox Range("B2")
```

	A	B	C
1	1 A		
2	2 B		
3	3 C		
4			
5			

Affichage d'un texte et de la valeur d'une variable

Remarquez qu'ici, la variable est numérique, et que VBA accepte qu'on la concatène à une chaîne avec l'opérateur de concaténation « & » :

```
Dim n as Integer  
n =  
12
```

```
MsgBox "n vaut " & n
```

	A	B	C
1	1 A		
2	2 B		
3	3 C		
4			
5			

Affichage d'un texte sur plusieurs lignes

Le passage à la ligne se fait grâce au caractère **Chr(10)** ou à la constante **vbLf**. Remarquez également le caractère de soulignement () en fin de ligne qui permet de répartir l'instruction sur plusieurs lignes pour améliorer sa lisibilité :

```
MsgBox "Un message sur plusieurs lignes." & Chr(10) & _
```

```
"Le passage à la ligne se fait avec un Chr(10)" & vbLf & _  
"ou avec la constante vbLf"
```



Définition du titre de la MsgBox()

Pour modifier le texte qui apparait dans la barre de titre d'une **MsgBox()**, vous devez passer trois paramètres à la fonction. Voici un exemple :

```
MsgBox "Texte dans la MsgBox()", , "Titre de la MsgBox()"
```



Ici, le deuxième paramètre est vide, ce qui provoque l'affichage d'une boîte de dialogue standard. Vous pourriez également utiliser les constantes suivantes :

vbCritical



vbQuestion



vbExclamation



vbInformation



Voici un exemple :

```
MsgBox "Texte dans la MsgBox()", vbExclamation, "Titre de la MsgBox()"
```



La fonction MsgBox()

MsgBox() peut également être utilisé en tant que fonction. Dans ce cas, le deuxième argument indique le nombre et la nature des boutons affichés dans la boîte de dialogue :

Constante	Effet
vbOKOnly	OK
vbOKCancel	OK Annuler
vbAbortRetryIgnore	Abandonner Recommencer Ignorer
vbYesNoCancel	Oui Non Annuler
vbYesNo	Oui Non
vbRetryCancel	Recommencer Annuler

Lorsqu'une boîte **MsgBox()** comporte plusieurs boutons, vous pouvez choisir celui qui est actif par défaut. L'utilisateur pourra appuyer sur la touche *Entrée* du clavier pour simuler un clic sur ce bouton. Pour cela, vous utiliserez les constantes du tableau suivant :

Constante	Effet
vbDefaultButton1	Bouton 1 par défaut
vbDefaultButton2	Bouton 2 par défaut
vbDefaultButton3	Bouton 3 par défaut

Vous pouvez également utiliser les constantes **vbCritical**, **vbQuestion**, **vbExclamation** et **vbInformation** pour ajouter une icône dans la boîte de dialogue.

Mais alors, si vous choisissez (par exemple) les boutons de la boîte de dialogue, comment indiquer en plus quel bouton sera utilisé par défaut et quelle icône vous voulez utiliser ? Eh bien tout simplement en additionnant les constantes correspondantes.

Par exemple, pour afficher une boîte de dialogue qui :

- affiche le texte « Voulez-vous continuer » ;

- contient les boutons Oui (sélectionné par défaut) et Non ;
- affiche une icône vbQuestion.

Vous utiliserez ces instructions :

```
MsgBox("Voulez-vous continuer ?", vbYesNo + vbDefaultButton1 + vbQuestion,
"Important")
```

Mais attention : ici, **MsgBox()** est utilisé en tant que fonction et non en tant que méthode. Elle retourne donc une valeur qui doit être affectée à une variable ou directement testée. Les valeurs retournées par la fonction **MsgBox()** peuvent être les suivantes :

Valeur renvoyée	Signification
vbOK	Bouton OK cliqué
vbCancel	Bouton Annuler cliqué
vbAbort	Bouton Abandonner cliqué
vbRetry	Bouton Recommencer cliqué
vbIgnore	Bouton Ignorer cliqué
vbYes	Bouton Oui cliqué
vbNo	Bouton Non cliqué

Voici un exemple de code :

```
If MsgBox("Voulez-vous continuer ?", vbYesNo + vbDefaultButton1 +
vbQuestion, "Important") = vbYes Then

    MsgBox ("OK, on continue")

Else

    MsgBox ("C'est d'accord, on s'arrête ")

End If
```

Voici le résultat :



Le bouton **Oui** est sélectionné par défaut. Si l'utilisateur appuie sur la touche *Entrée* du clavier ou clique sur le bouton **Oui**, une boîte de dialogue s'affiche :



S'il clique sur **Non**, une autre boîte de dialogue s'affiche :



La fonction InputBox()

Cette section ne serait pas complet si la fonction **InputBox()** n'était pas citée. Cette fonction demande à l'utilisateur de saisir une information textuelle ou numérique. Voici sa syntaxe :

```
InputBox ( message [, titre ] [, défaut ] [, xpos ] [, ypos ] )
```

Où :

- **message** représente le message à afficher dans la boîte de dialogue ;
- **titre** représente le texte affiché dans la barre de titre (ce paramètre est optionnel) ;
- **défaut** représente la valeur par défaut (ce paramètre est optionnel) ;
- **xpos** et **ypos** représentent les coordonnées x et y de l'angle supérieur gauche de l'InputBox par rapport à l'angle supérieur gauche de l'écran (ces paramètres sont optionnels).

Voici un exemple de code :

```
Dim prenom As String

While Len(prenom) = 0

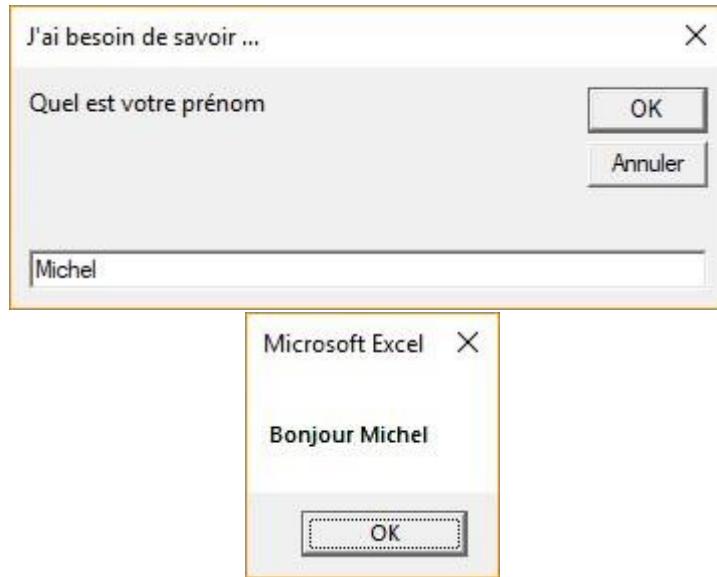
    prenom = InputBox("Quel est votre prénom", "J'ai besoin de savoir")

Wend

MsgBox "Bonjour " & prenom
```

Ici, une boucle **While Wend** répète autant de fois que nécessaire la saisie du prénom jusqu'à ce que l'utilisateur entre son prénom (c'est-à-dire une chaîne non vide).

Voici un exemple d'exécution :



Remarque

Si vous affectez le résultat de la fonction **InputBox()** à une variable numérique et que l'utilisateur saisit du texte, une erreur est générée et le programme prend fin, à moins que vous ne mettiez en place un gestionnaire d'erreurs.

Traitement du texte

Comme vous allez le voir dans cette section, VBA possède de nombreuses fonctions pour manipuler du texte. Au fil de l'article, vous découvrirez les fonctions **Len()**, **Left()**, **Right()**, **Mid()**, **LTrim()**, **RTrim()**, **Trim()**, **UCase()**, **LCase()**, **Proper()**, **Replace()**, **Val()**, **IsNumeric()**, **IsDate()** et beaucoup d'autres.

Longueur d'une chaîne

La fonction **Len()** retourne la longueur de la chaîne passée en argument.

Exemple :

```
Dim ch As String
ch = "Ceci est une chaîne"
MsgBox "Longueur de la chaîne : " & Len(ch)
```

Extraction de sous-chaînes

Pour extraire une sous-chaîne d'une chaîne, vous pouvez utiliser les fonctions **Left()**, **Right()** et **Mid()** :

- La fonction **Left(ch, nombre)** renvoie les nombre caractères au début de la chaîne **ch**.
 - La fonction **Right(ch, nombre)** renvoie les nombre caractères à la fin de la chaîne **ch**.
- Enfin, la fonction **Mid(ch, deb, long)** renvoie les long caractères de la chaîne **ch**, en partant du caractère de position **deb**.

Exemples :

```
Dim s As String

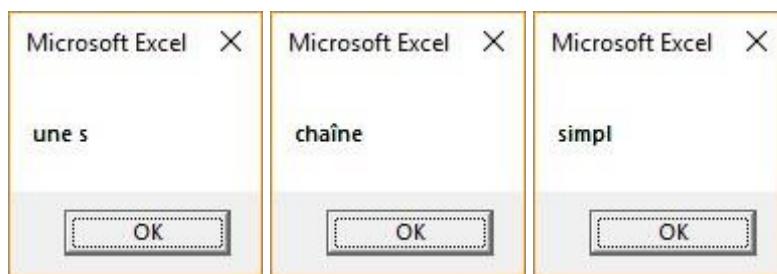
s = "une simple chaîne"

MsgBox Left(s, 5)

MsgBox Right(s, 6)

MsgBox Mid(s, 5, 5)
```

Voici les boîtes de dialogue affichées par ce code :



Suppression d'espaces au début et/ou à la fin

Les fonctions **LTrim()**, **RTrim()** et **Trim()** permettent de supprimer les espaces (respectivement) au début, à la fin et des deux côtés d'une chaîne.

Exemples :

```
Dim s As String

s = "      >une chaîne<      "

MsgBox "LTrim(s) : " & LTrim(s) & " longueur " & Len(LTrim(s)) & vbCrLf & _
"RTrim(s) : " & RTrim(s) & " longueur " & Len(RTrim(s)) & vbCrLf & _
"Trim(s) : " & Trim(s) & " longueur " & Len(Trim(s))
```

Voici le résultat de l'exécution de ce code :



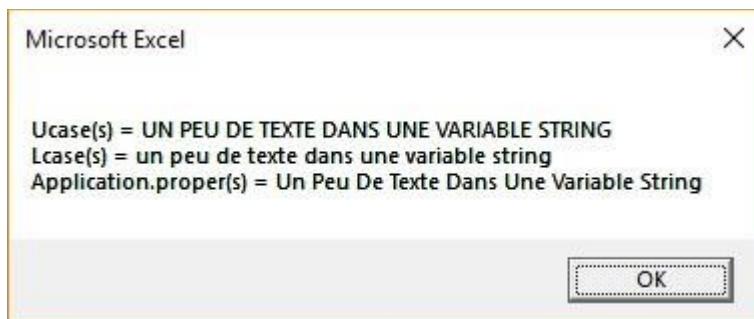
Casse des caractères

Les fonctions **Ucase()**, **Lcase()** et **Application.Proper()** permettent de modifier la casse des caractères.

Voici un exemple d'utilisation :

```
Dim s As String  
  
s = "un peu de texte dans une variable String"  
  
MsgBox "Ucase(s) = " & UCase(s) & vbCrLf & _  
       "Lcase(s) = " & LCase(s) & vbCrLf & _  
       "Application.proper(s) = " & Application.Proper(s)
```

Voici le résultatat de ce code :



Valeur numérique d'une chaîne

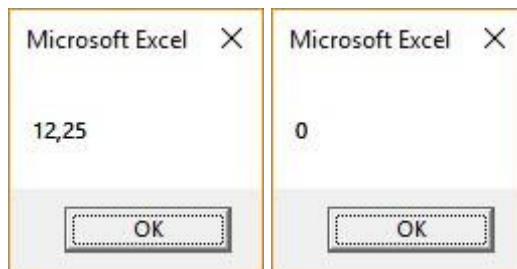
La fonction **Val()** convertit son argument chaîne en un nombre lorsque cela est possible. Lorsque cela n'est pas possible, elle retourne la valeur **0**.

Voici un exemple de code :

```
Dim s As String  
  
Dim r As Single  
  
s = "12.25"  
r = Val(s)  
MsgBox r  
s = "bonjour"
```

```
r =  
Val(s)  
MsgBox r
```

Et le résultat de son exécution :



Remplacement d'une sous-chaîne par une autre

La fonction **Replace()** permet de remplacer une sous-chaîne par une autre dans une chaîne.

Voici un exemple de code :

```
Dim s As String  
  
s = "Illustration de la fonction Replace()"  
  
MsgBox Replace(s, "e", "E")
```

Ici, tous les « e » sont remplacés par des « E ». Voici le résultat :



Deux autres paramètres peuvent être passés à la fonction **Replace()** : la position du premier remplacement et le nombre de remplacements. Voici un exemple pour mieux comprendre comment utiliser ces paramètres :

```
Dim s As String  
s = "la la la la  
lère"  
  
MsgBox Replace(s, "la", "LA", 2, 2)
```

Ici, deux remplacements seront effectués (le dernier paramètre) et la chaîne retournée commencera au deuxième caractère de la chaîne originale. Voici le résultat :



Conversions et mises en forme

Plusieurs fonctions permettent de convertir une donnée d'un certain type dans un autre type :

Fonction	Conversion en
CBool()	Boolean
CByte()	Byte
CCur()	Currency
CDate()	Date
CDbl()	Double
CDec()	Decimal
CInt()	Integer
CLng()	Long
CSng()	Single
CStr()	String
CVar()	Variant

Par exemple, pour convertir un **Single** en **Integer**, vous pourriez utiliser ces instructions :

```
Dim pi As Single  
pi = 3.14158265  
MsgBox CInt(pi)
```

Voici le résultat :



Mise en forme

La fonction **Format()** permet de mettre en forme des nombres, des dates et des chaînes. Elle retourne une valeur de type **String**. Voici sa syntaxe :

```
Format(expression, format)
```

Où **expression** est l'expression à mettre en forme et **format** est le format de mise en forme à appliquer à l'expression.

Voici quelques exemples :

```
Dim s As Single
```

```
s = 1234.456
```

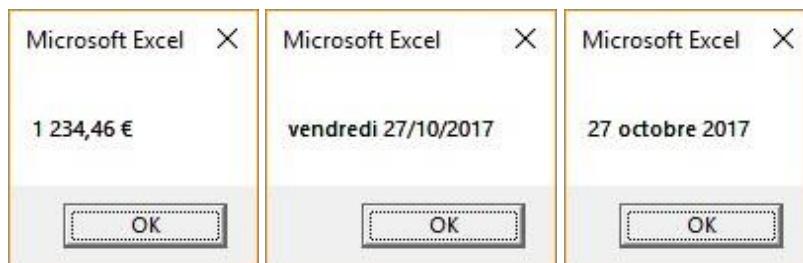
```
MsgBox Format(s, "0 000.00 €")
```

```
Dim d As Date  
d =  
Date
```

```
MsgBox Format(d, "dddd dd/mm/yyyy")
```

```
MsgBox Format(d, "dd mmmm yyyy")
```

Et voici le résultat de l'exécution :



Pour aller plus loin avec la fonction **Format()**, je vous suggère de consulter [la page dédiée à cette fonction sur MSDN](#).

Comme vous allez le voir dans cette section, VBA possède de nombreuses fonctions pour manipuler du texte. Au fil de l'article, vous découvrirez les fonctions **Len()**, **Left()**, **Right()**, **Mid()**, **LTrim()**, **RTrim()**, **Trim()**, **UCase()**, **LCase()**, **Proper()**, **Replace()**, **Val()**, **IsNumeric()**, **IsDate()** et beaucoup d'autres.

Longueur d'une chaîne

La fonction **Len()** retourne la longueur de la chaîne passée en argument.

Exemple :

```
Dim ch As String  
ch = "Ceci est une  
chaîne"
```

```
MsgBox "Longueur de la chaîne : " & Len(ch)
```

Extraction de sous-chaînes

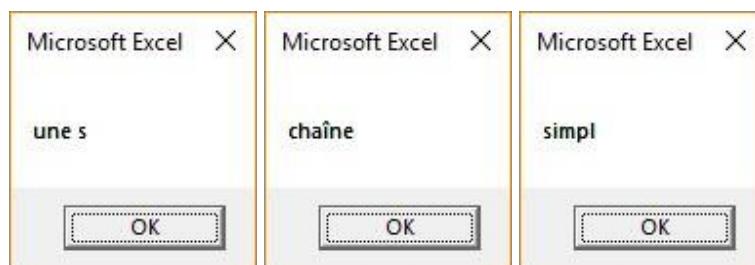
Pour extraire une sous-chaîne d'une chaîne, vous pouvez utiliser les fonctions **Left()**, **Right()** et **Mid()** :

- La fonction **Left(ch, nombre)** renvoie les nombre caractères au début de la chaîne **ch**.
 - La fonction **Right(ch, nombre)** renvoie les nombre caractères à la fin de la chaîne **ch**.
- Enfin, la fonction **Mid(ch, deb, long)** renvoie les long caractères de la chaîne **ch**, en partant du caractère de position **deb**.

Exemples :

```
Dim s As String  
  
s = "une simple chaîne"  
  
MsgBox Left(s, 5)  
  
MsgBox Right(s, 6)  
  
MsgBox Mid(s, 5, 5)
```

Voici les boîtes de dialogue affichées par ce code :



Suppression d'espaces au début et/ou à la fin

Les fonctions **LTrim()**, **RTrim()** et **Trim()** permettent de supprimer les espaces (respectivement) au début, à la fin et des deux côtés d'une chaîne.

Exemples :

```
Dim s As String  
  
s = "      >une chaîne<      "  
  
MsgBox "LTrim(s) : " & LTrim(s) & " longueur " & Len(LTrim(s)) & vbCrLf & _  
"RTrim(s) : " & RTrim(s) & " longueur " & Len(RTrim(s)) & vbCrLf & _  
"Trim(s) : " & Trim(s) & " longueur " & Len(Trim(s))
```

Voici le résultat de l'exécution de ce code :



Casse des caractères

Les fonctions **Ucase()**, **Lcase()** et **Application.Proper()** permettent de modifier la casse des caractères.

Voici un exemple d'utilisation :

```
Dim s As String  
  
s = "un peu de texte dans une variable String"  
  
MsgBox "Ucase(s) = " & UCase(s) & vbCrLf & _  
"Lcase(s) = " & LCase(s) & vbCrLf & _  
"Application.proper(s) = " & Application.Proper(s)
```

Voici le résultat de ce code :



Valeur numérique d'une chaîne

La fonction **Val()** convertit son argument chaîne en un nombre lorsque cela est possible. Lorsque cela n'est pas possible, elle retourne la valeur **0**.

Voici un exemple de code :

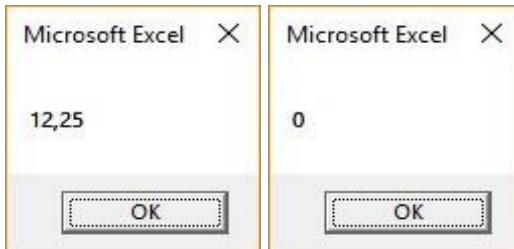
```
Dim s As String  
  
Dim r As Single  
  
s = "12.25"  
r =  
Val(s)
```

```

MsgBox r
s = "bonjour"
r =
Val(s)
 MsgBox r

```

Et le résultat de son exécution :



Remplacement d'une sous-chaîne par une autre

La fonction **Replace()** permet de remplacer une sous-chaîne par une autre dans une chaîne.

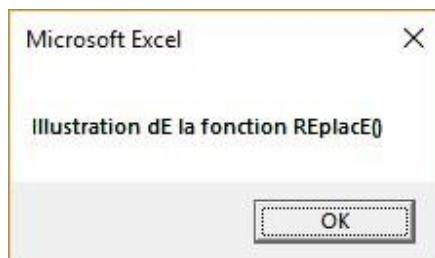
Voici un exemple de code :

```

Dim s As String
s = "Illustration de la fonction Replace()"
MsgBox Replace(s, "e", "E")

```

Ici, tous les « e » sont remplacés par des « E ». Voici le résultat :



Deux autres paramètres peuvent être passés à la fonction **Replace()** : la position du premier remplacement et le nombre de remplacements. Voici un exemple pour mieux comprendre comment utiliser ces paramètres :

```

Dim s As String
s = "la la la la lère"
MsgBox Replace(s, "la", "LA", 2, 2)

```

Ici, deux remplacements seront effectués (le dernier paramètre) et la chaîne retournée commencera au deuxième caractère de la chaîne originale. Voici le résultat :



Conversions et mises en forme

Plusieurs fonctions permettent de convertir une donnée d'un certain type dans un autre type :

Fonction	Conversion en
CBool()	Boolean
CByte()	Byte
CCur()	Currency
CDate()	Date
CDbl()	Double
CDec()	Decimal
CInt()	Integer
CLng()	Long
CSng()	Single
CStr()	String
CVar()	Variant

Par exemple, pour convertir un **Single** en **Integer**, vous pourriez utiliser ces instructions :

```
Dim pi As Single  
  
pi = 3.14158265  
  
MsgBox CInt(pi)
```

Voici le résultat :



Mise en forme

La fonction **Format()** permet de mettre en forme des nombres, des dates et des chaînes. Elle retourne une valeur de type **String**. Voici sa syntaxe :

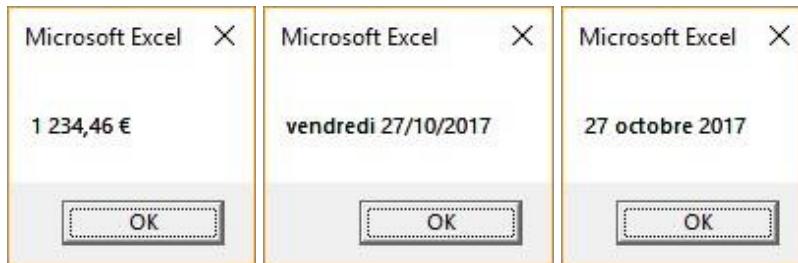
```
Format(expression, format)
```

Où **expression** est l'expression à mettre en forme et **format** est le format de mise en forme à appliquer à l'expression.

Voici quelques exemples :

```
Dim s As Single  
s = 1234.456  
MsgBox Format(s, "0 000.00 €")  
  
Dim d As Date  
d = Date  
  
MsgBox Format(d, "dddd dd/mm/yyyy")  
MsgBox Format(d, "dd mmmm yyyy")
```

Et voici le résultat de l'exécution :



Pour aller plus loin avec la fonction **Format()**, je vous suggère de consulter [la page dédiée à cette fonction sur MSDN](#).

Traitement des nombres

Cette section regroupe les opérateurs et les fonctions dédiés à la manipulation des nombres en VBA Excel.

Opérateurs mathématiques

Les opérateurs mathématiques sont très classiques. Notez cependant l'opérateur de division entière (\) et le module (Mod) qui ne se retrouvent pas dans tous les langages :

Opérateur	Signification
$^$	Mise à la puissance
$+$	Addition
$-$	Soustraction
$*$	Multiplication
$/$	Division
\backslash	Division entière
Mod	Modulo

Exemples :

```

Dim n1, n2 As Single
n1 =
13.34   n2
= 3.46

Debug.Print "n1 = " & n1

Debug.Print "n2 = " & n2

Debug.Print "n1^3 = " & n1 ^ 3

Debug.Print "n1/n2 = " & n1 / n2

Debug.Print "n1\nn2 = " & n1 \ n2

Debug.Print "n1 Mod 2 = " & n1 Mod n2

```

Voici le résultat de ce code :

```

n1 = 13,34
n2 = 3,46
n1^3 = 2373,927704

n1/n2 = 3,85549128697245
n1\nn2 = 4
n1 Mod 2 = 1

```

Opérateurs de comparaison

Les opérateurs de comparaison sont très classiques :

Opérateur	Signification
$=$	Egale à
$<$	Inférieur à
\leq	Inférieur ou égal à
$>$	Supérieur à

<code>>=</code>	Supérieur ou égal à
<code><></code>	Different de

Opérateurs logiques

Les opérateurs logiques sont également très classiques :

Opérateur	Signification
And	Et logique
Or	Ou logique
Not	Non logique

Fonctions mathématiques

Le langage VBA contient quelques fonctions mathématiques dédiées à la manipulation des nombres. Elles sont résumées dans le tableau suivant :

Fonction	Valeur renournée
Abs()	Valeur absolue de l'argument
Atn()	Double qui représente l'arc tangente de l'argument
Cos()	Double qui représente le cosinus de l'angle passé en argument
Exp()	Double qui représente la base de logarithmes népériens mise à la puissance de l'argument
Fix()	Partie entière de l'argument
Hex()	String qui représente la valeur hexadécimale de l'argument
Int()	Partie entière de l'argument
Log()	Double qui indique le logarithme népérien de l'argument
Oct()	Variant (String) qui représente la valeur octale de l'argument
Rnd()	Single qui représente un nombre aléatoire
Round()	Argument arrondi à un nombre spécifié de décimales
Sgn()	Variant (Integer) qui indique le signe de l'argument
Sin()	Double qui représente le sinus de l'angle passé en argument
Sqr()	Double qui représente la racine carrée du nombre passé en argument
Tan()	Double qui représente la tangente de l'angle passé en argument
Val()	Evaluation numérique de l'argument chaîne

Fonctions de conversion

Et pour terminer, voici quelques fonctions de conversion :

Fonction	Valeur rentrée	Exemple
CBool()	Evalue l'argument et retourne Vrai ou Faux en conséquence retourne Vrai	CBool(1<2)
CByte()	Retourne la conversion en Byte (entre 0 et 255) de l'argument	CByte(2.35) retourne le Byte 2
CCur()	Retourne la conversion en Currency de CByte(12345.678989) retourne le l'argument	Currency 12345.679
CDbl()	Retourne la conversion en Double de l'argument	CDbl(12.456789) retourne le Double 12.456789

Mise en forme

La fonction **Format()** permet de mettre en forme des nombres. Elle retourne une valeur de type **String**. Voici sa syntaxe :

Format(expression, format)

Où **expression** est l'expression à mettre en forme et **format** est le format de mise en forme à appliquer à l'expression.

Voici quelques exemples :

```
Dim s As Single
```

```
s = 1234.456
```

```
MsgBox Format(s, "0 000.00 €")
```

```
s = 1234567
```

```
MsgBox Format(s, "0.00")
```

```
MsgBox Format(s, "#,##0")
```

Voici le résultat de l'exécution :

Pour aller plus loin avec la fonction **Format()**, je vous suggère de consulter [la page dédiée à cette fonction sur MSDN](#).



Traitement des dates

Fonctions de base dédiées aux dates

Il existe un grand nombre de fonctions dédiées à la manipulation des dates en VBA. Tant mieux, car sans elles, vous devriez écrire beaucoup plus de lignes de code lorsque vous manipulez des dates et heures...

Fonction	Valeur renournée par la fonction
Date	Variant (Date) qui contient la date système
Time	Variant (Date) qui contient l'heure système
Now	Variant (Date) qui contient la date et l'heure système
Day	Variant (Integer) qui représente le jour du mois de la date passée en argument
Month	Variant (Integer) qui représente le mois de la date passée en argument
MonthName	Variant (String) qui représente le nom du mois passé en argument
Year	Variant (Integer) qui représente l'année de la date passée en argument
Weekday	Variant (Integer) qui représente le jour de la semaine
WeekdayName	Variant (String) qui représente le nom du jour de la semaine passé en argument
Hour	Variant (Integer) qui représente les heures de la valeur passée en argument
Minute	Variant (Integer) qui représente les minutes de la valeur passée en argument
Second	Variant (Integer) qui représente les secondes de la valeur passée en argument
Timer	Single qui représente le nombre de secondes écoulées depuis minuit

Voici quelques exemples d'utilisation de ces fonctions :

```
Debug.Print Date
Debug.Print Time
Debug.Print Now
Debug.Print Day(Date)
```

```
Debug.Print Month(Date)  
Debug.Print MonthName(Month(Date))  
Debug.Print Year(Date)  
Debug.Print Weekday(Date)  
Debug.Print WeekdayName(Weekday(Date) - 1)  
Debug.Print Hour(Now)  
Debug.Print Minute(Now)  
Debug.Print Second(Now)  
Debug.Print Timer
```

Et voici les résultats affichés dans la fenêtre **Exécution** :

02/01/2025

17:19:41

02/01/2025 17:19:41

30

02
janvier
2025

30
lundi
2024

19

41

62381,12

Les fonctions IsDate() et CDate()

La fonction IsDate(d) retourne la valeur :

- Vrai si l'argument est une date ;
- Faux si l'argument n'est pas une date.

La fonction CDate() transforme l'argument chaîne qui lui est passé en un date Exemples :

```
Debug.Print "IsDate(""12/8/1960"" : " & IsDate("12/08/1960")  
Debug.Print "IsDate(""122/08/1960"" : " & IsDate("122/08/1960")
```

```

Dim d As Variant

Debug.Print "d est défini en tant que variant"

Debug.Print "IsDate(d) : " & IsDate(d)

d = CDate("22/12/2024")

Debug.Print "Après cette instruction :"

Debug.Print "d = CDate(""22/12/2024"")"

Debug.Print "IsDate(d) vaut " & IsDate(d)

```

Voici le résultat de ce code :

```

IsDate("12/8/1960") : Vrai

IsDate("122/08/1960") : Faux

d est défini en tant que variant

IsDate(d) : Faux

```

Après cette instruction :

```
d = CDate("22/12/2024")
```

IsDate(d) vaut Vrai

Les fonctions de date qui travaillent sur des intervalles

Plusieurs fonctions vous permettent de travailler simplement sur des intervalles. Par exemple pour ajouter 25 jours à une date ou encore pour soustraire 3 mois à une date. Les intervalles sont codés sous la forme de String. Les différentes valeurs possibles sont résumées dans ce tableau :

Intervalle	Effet
yyyy	Année (entre 100 et 9999)
y	Jour de l'année (entre 1 et 365)
m	Mois (entre 1 et 12)
q	Trimestre (entre 1 et 4)
ww	Semaine (entre 1 et 53)
w	Jour (entre 1 et 7)
d	Jour (entre 1 et 31)
h	Heure (entre 0 et 23)
n	Minute (entre 0 et 59)
s	Seconde (entre 0 et 59)

La fonction DateAdd()

La fonction DateAdd() retourne un Variant (Date) qui contenant une date à laquelle un intervalle de temps spécifié a été ajouté.

```
DateAdd(intervalle, nombre, date)
```

Où :

- intervalle est un des codes du tableau précédent ;
- nombre est le nombre d'intervalles à ajouter à la date de référence, spécifiée dans le troisième argument.

Exemples :

```
Debug.Print "Date système : " & Date  
  
Debug.Print "Dans 3 mois : " & DateAdd("m", 3, Date)  
Debug.Print "Il y a 3 mois : " & DateAdd("m", -3, Date)  
Debug.Print "Dans 5 jours : " & DateAdd("y", 5, Date)  
  
Debug.Print "Il y a 1 trimestre : " & DateAdd("q", -1, Date)  
  
Debug.Print "Heure système : " & Time  
  
Debug.Print "Il y a 2h20 : " & DateAdd("n", -140, Time)  
Debug.Print "Dans 2 ans : " & DateAdd("yyyy", 2, Now)
```

Voici le résultat :

```
Date système : 02/01/2025  
  
Dans 3 mois : 02/01/2025  
  
Il y a 3 mois : 02/10/2024  
  
Dans 5 jours : 07/01/2025  
  
Il y a 1 trimestre : 02/10/2024  
  
Heure système : 17:51:39  
  
Il y a 2h20 : 15:31:39  
  
Dans 2 ans : 02/01/2023 17:51:39
```

La fonction DateDiff()

La fonction DateDiff() retourne un Variant (Long) qui indique le nombre d'intervalles de temps entre deux dates données. Voici sa syntaxe :

```
DateDiff(intervalle, date1, date2)
```

Où **intervalle** est l'intervalle de temps (voir le tableau au début de la section « Les fonctions de date qui travaillent sur des intervalles ») qui sépare les dates **date1** et **date2**.

Exemples :

```
Dim d1, d2 As Variant  
  
d1 = "20/12/2016"  
  
d2 = "15/06/2020"  
  
Debug.Print "Les deux dates à comparer : " & d1 & " et " & d2  
  
Debug.Print "Nb de mois entre ces deux dates : " & DateDiff("m", d1, d2)  
Debug.Print "Nb de semaines entre ces deux dates : " & DateDiff("ww", d1, d2)  
  
Debug.Print "Nb de jours entre ces deux dates : " & DateDiff("w", d1, d2)  
Dim h1, h2 As Variant  
  
h1 = "5:10:12"  
  
h2 = "17:30:45"  
  
Debug.Print "Les deux heures à comparer h1 et h2 : " & h1 & " et " & h2  
Debug.Print "Nb d'heures entre h1 et h2 : " & DateDiff("h", h1, h2)  
Debug.Print "Nb de minutes entre h1 et h2 : " & DateDiff("n", h1, h2)  
Debug.Print "Nb de secondes entre h1 et h2 : " & DateDiff("s", h1, h2)
```

Voici le résultat de ce code :

Les deux dates à comparer : 20/12/2016 et 15/06/2020

Nb de mois entre ces deux dates : 42

Nb de semaines entre ces deux dates : 182

Nb de jours entre ces deux dates : 181

Les deux heures à comparer h1 et h2 : 5:10:12 et 17:30:45

Nb d'heures entre h1 et h2 : 12

Nb de minutes entre h1 et h2 : 740

Nb de secondes entre h1 et h2 : 44433

La fonction DatePart()

La fonction DatePart() retourne un Variant (Integer) qui représente l'année, le trimestre, le mois, la semaine ou le jour d'une date. Voici sa syntaxe :

```
DatePart (partie, date)
```

Où **partie** représente l'élément à extraire de la date spécifiée en deuxième argument.

Exemples :

```
Debug.Print "Date système : " & Date  
  
Debug.Print "Semaine de la date système : " & DatePart("ww", Date)  
Debug.Print "Trimestre de la date système : " & DatePart("q", Date)  
Debug.Print "Jour de la semaine de la date système : " & DatePart("w",  
Date) - 1
```

Voici le résultatat de ce code :

```
Date système : 30/10/2017  
  
Semaine de la date système : 44  
  
Trimestre de la date système : 4  
  
Jour de la semaine de la date système : 1
```

Les fonctions DateSerial() et TimeSerial()

La fonction **DateSerial()** retourne un Variant (Date) qui correspond à l'année, le mois et le jour passés en argument. Voici sa syntaxe :

```
DateSerial(annee, mois, jour)
```

Où **annee**, **mois** et **jour** sont (respectivement) l'année, le mois et le jour de la date à reconstituer.

La fonction DateSerial() retourne un Variant (Date) qui correspond à l'année, le mois et le jour passés en argument. Voici sa syntaxe :

```
DateSerial(annee, mois, jour)
```

Où **annee**, **mois** et **jour** sont (respectivement) l'année, le mois et le jour de la date à reconstituer.

La fonction **TimeSerial()** retourne un Variant (Date) qui correspond aux heures, minutes et secondes passées en argument. Voici sa syntaxe :

```
TimeSerial(heures, minutes, secondes)
```

Où **heures**, **minutes** et **secondes** sont (respectivement) les heures, les minutes et les secondes de l'heure à reconstituer.

Exemples :

```
Debug.Print DateSerial(2018, 10, 25)
```

```
Debug.Print TimeSerial(22, 15, 12)
```

Voici le résultat de ce code :

25/10/2018

22:15:12

DateValue Renvoie une valeur de type Variant (Date).

TimeValue Renvoie une valeur de type Variant (Date) contenant une heure.

Mise en forme

La fonction **Format()** permet de mettre en forme des dates. Elle retourne une valeur de type **String**. Voici sa syntaxe :

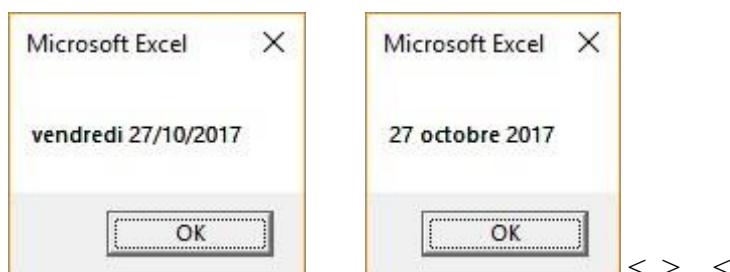
```
Format(expression, format)
```

Où **expression** est l'expression à mettre en forme et **format** est le format de mise en forme à appliquer à l'expression.

Voici quelques exemples :

```
Dim d As Date  
d =  
Date  
  
MsgBox Format(d, "dddd dd/mm/yyyy")  
  
MsgBox Format(d, "dd mmmm yyyy")
```

Voici le résultat de l'exécution :



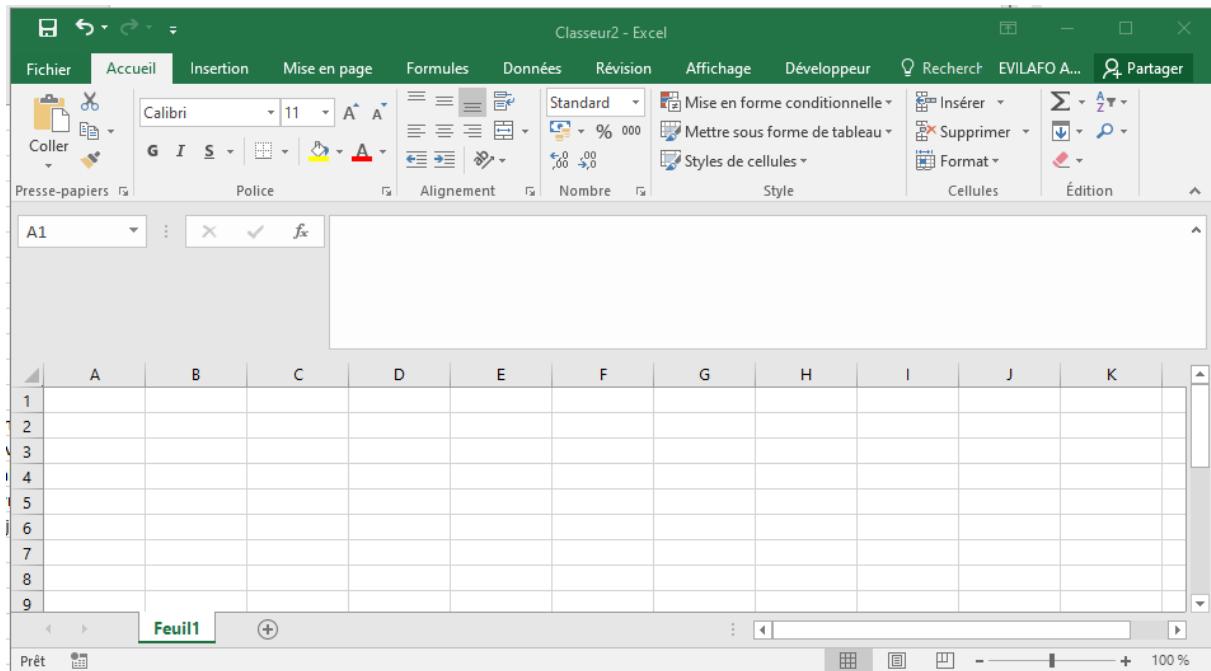
Pour aller plus loin avec la fonction **Format()**, je vous suggère de consulter [la page dédiée à cette fonction sur MSDN](#).

Traitement des classeurs

Le langage VBA permet d'ouvrir, de modifier et de sauvegarder des classeurs. Cette section va vous montrer comment.

Ouvrir un classeur vierge

Il est très simple d'ouvrir un nouveau classeur vierge avec VBA. Pour cela, vous utiliserez la méthode Workbooks.Add. Sans argument, cette méthode ouvre un nouveau classeur nommé Classeur1 (si c'est le premier), classeur2.xlsx (si c'est le deuxième), etc. :



Si nécessaire, vous pouvez donner un nom à ce nouveau classeur et le sauvegarder dans un dossier quelconque :

```
Set nouveau = Workbooks.Add  
nouveau.SaveAs  
Filename:="c:\data\classeurs\nouveau.xlsx"
```

Remarquez l'utilisation du mot clé **Set** (et non Dim) pour définir la variable objet **nouveau** dans laquelle est stocké le nouveau classeur.

La propriété **SaveAs** définit le dossier et le nom du classeur et le sauvegarde.

Ouvrir un classeur

Dans un précédent article, vous avez vu qu'il était possible de s'adresser à un classeur nommé classeur1.xlsm avec l'instruction :

```
Workbooks("classeur1.xlsx")
```

Ou encore à la cellule C8 de la feuille Feuil1 du classeur Classeur1.xlsx avec cette instruction :

```
Workbooks("classeur1.xlsx").Worksheets("Feuil1").Range("C8")
```

Vous allez maintenant apprendre à ouvrir un classeur avec l'instruction **Workbooks.Open** dont voici la syntaxe :

```
Workbooks.Open "classeur"
```

Où **classeur** représente le chemin complet du classeur que vous voulez ouvrir.

Après l'exécution de cette instruction, le classeur ouvert devient le classeur actif.

Ouverture d'un classeur avec un chemin explicite

Par exemple, pour ouvrir le classeur **société.xlsx** qui se trouve dans le dossier **data\classeurs** du disque **c:**, vous utiliserez cette instruction :

```
Workbooks.Open "c:\data\classeurs\société.xlsx" Ouverture
```

d'un classeur avec un chemin relatif

Dans l'instruction précédente, le chemin du classeur est indiqué de façon explicite. Si le classeur à ouvrir se trouve dans le même dossier que le classeur où le code VBA est exécuté, vous pouvez y faire référence de façon relative, c'est-à-dire sans préciser explicitement le chemin du dossier. Pour cela, vous utiliserez la propriété **Path** de l'objet **ActiveWorkbook** :

```
Dim chemin As String  
  
chemin = ActiveWorkbook.Path  
  
Workbooks.Open chemin & "\société.xlsx"
```

Fermer un classeur

Pour fermer un classeur, vous utiliserez l'instruction **Workbooks().Close** dont voici la syntaxe :

```
Workbooks("classeur").Close
```

Où **classeur** représente le nom du classeur que vous voulez fermer.

Par exemple, pour fermer le classeur **société.xlsx**, vous utiliserez cette instruction :

```
Workbooks("société.xlsx").Close
```

Si le classeur à fermer a été modifié depuis son ouverture, une boîte de dialogue vous demandera si vous voulez sauvegarder les modifications :

Pour éviter d'avoir à cliquer sur Enregistrer ou sur Ne pas enregistrer, vous pouvez choisir si le classeur doit ou ne doit pas être enregistré en ajoutant un paramètre à la commande de fermeture.

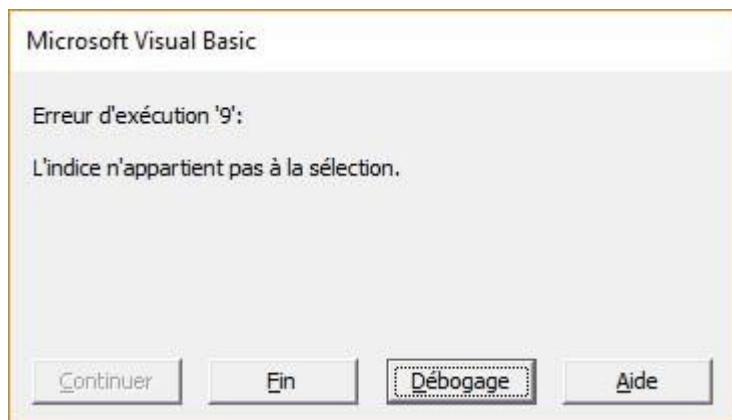
Pour sauvegarder par défaut le classeur avant de le fermer, ajoutez SaveChanges:=True à la suite de la commande de fermeture. Par exemple, pour sauvegarder puis fermer le classeur société.xlsm, vous utiliserez cette commande :

```
Workbooks("société.xlsm").Close SaveChanges:=True
```

Inversement, pour fermer un classeur sans sauvegarder les modifications, vous ajouterez SaveChanges:=False à la suite de la commande de fermeture. Par exemple, pour fermer le classeur société.xlsm sans sauvegarder les modifications, vous utiliserez cette commande :

```
Workbooks("société.xlsm").Close SaveChanges:=False
```

Pour terminer cette section sur la fermeture de classeurs, signalons qu'un message d'erreur s'affichera si vous tentez de fermer un classeur qui n'est pas ouvert :



Remarque

Vous voulez supprimer cette erreur ? Ajoutez cette instruction dans la procédure qui ferme le classeur :

```
On Error Resume Next
```

Pour aller plus loin sur la gestion des erreurs, consultez l'article intitulé » Gestion d'erreurs en VBA ».

Sauvegarder un classeur

En complément des commandes d'ouverture et de fermeture, sachez que vous pouvez également sauvegarder un classeur avec l'instruction **Workbooks().Save** dont voici la syntaxe :

```
Workbooks("classeur").Save
```

Où **classeur** représente le nom du classeur que vous voulez sauvegarder.

Notez qu'il est également possible de sauvegarder le classeur actif (celui qui a le focus) avec cette instruction :

```
ActiveWorkbook.Save
```

Si le classeur doit être sauvegardé dans un autre fichier, vous utiliserez l'instruction **Workbooks().SaveAs** :

```
Workbooks("classeur").SaveAs
```

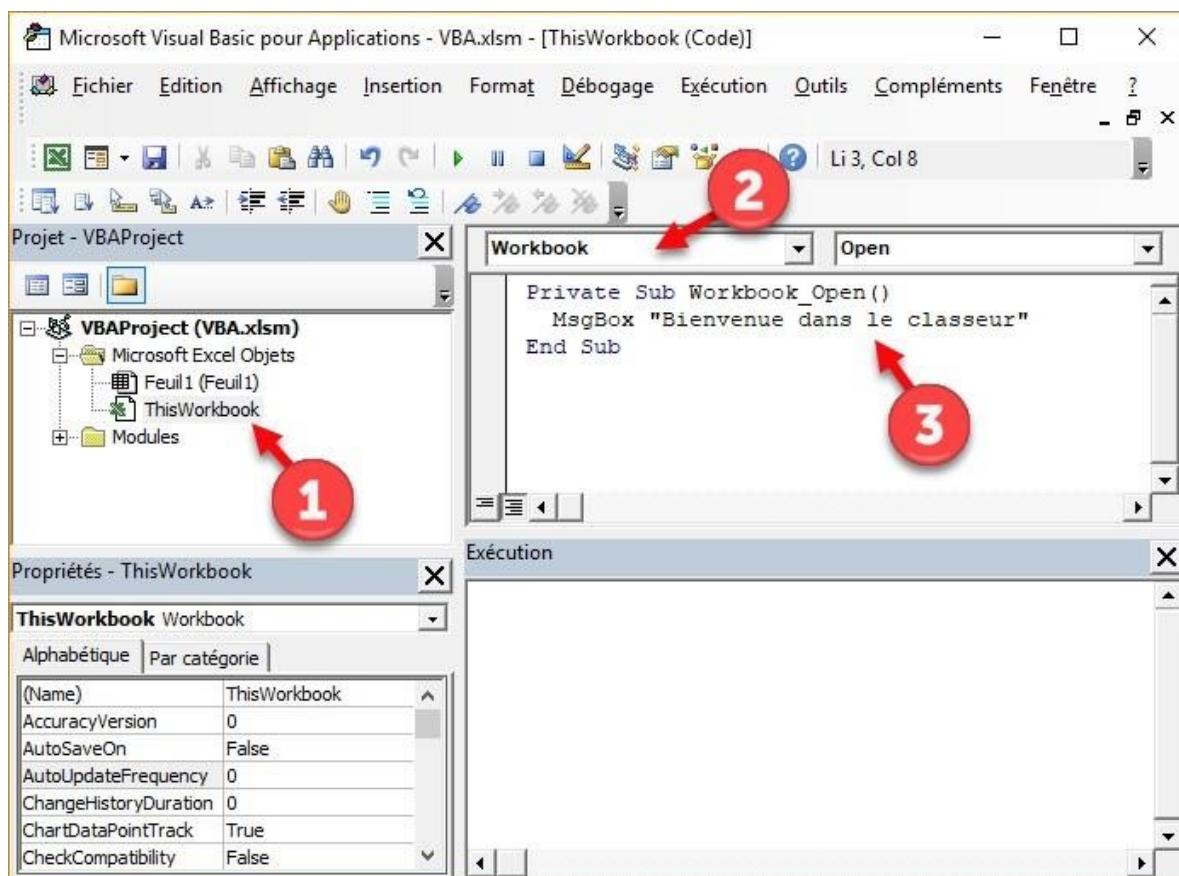
Où **classeur** représente le nom du classeur que vous voulez sauvegarder.

Exécuter une procédure à l'ouverture d'un classeur

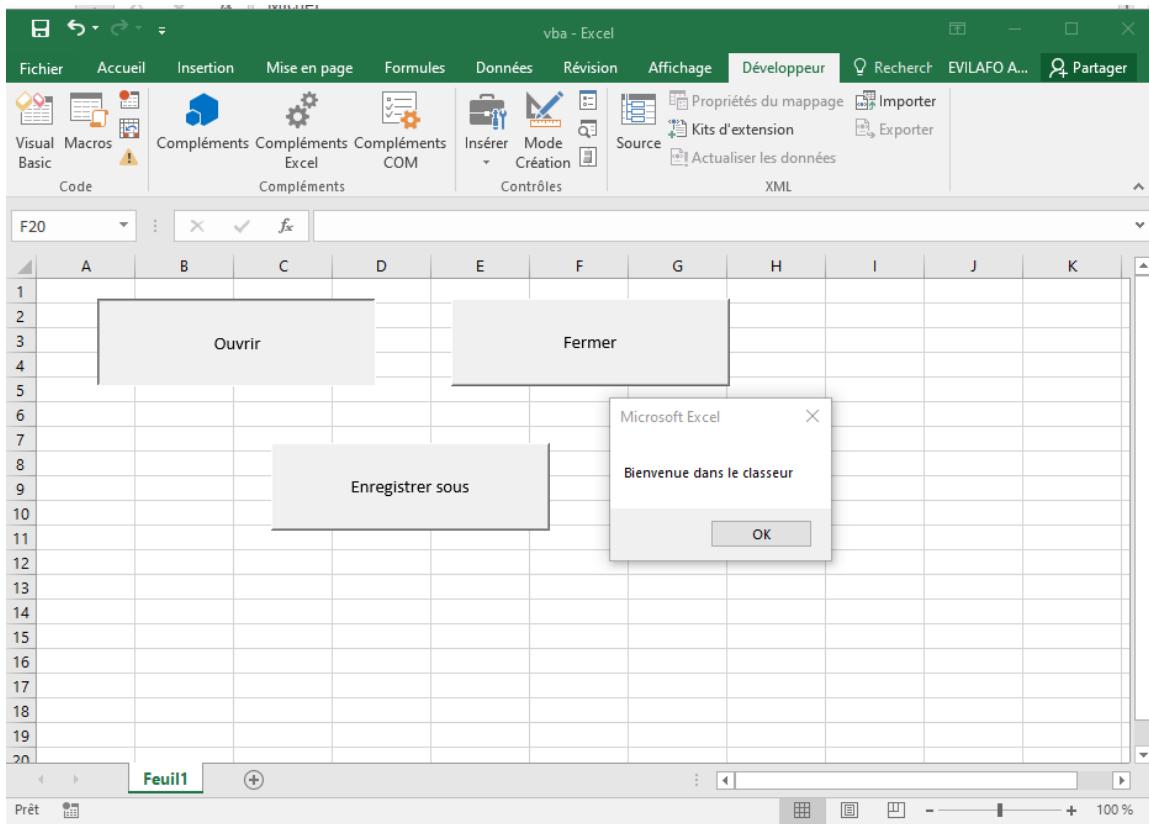
Il peut parfois être utile d'exécuter une macro à l'ouverture d'un classeur. Pour cela, vous devez créer la procédure **Workbook_Open()**.

Ouvrez la fenêtre Microsoft Visual Basic pour Applications du classeur concerné.

Doublecliquez sur **ThisWorkbook** dans la fenêtre **Projet** (1) et sélectionnez **Workbook** dans la liste déroulante **Objet** (2). La procédure **Workbook_Open()** est automatiquement créée. Il ne vous reste plus qu'à la compléter (3) :



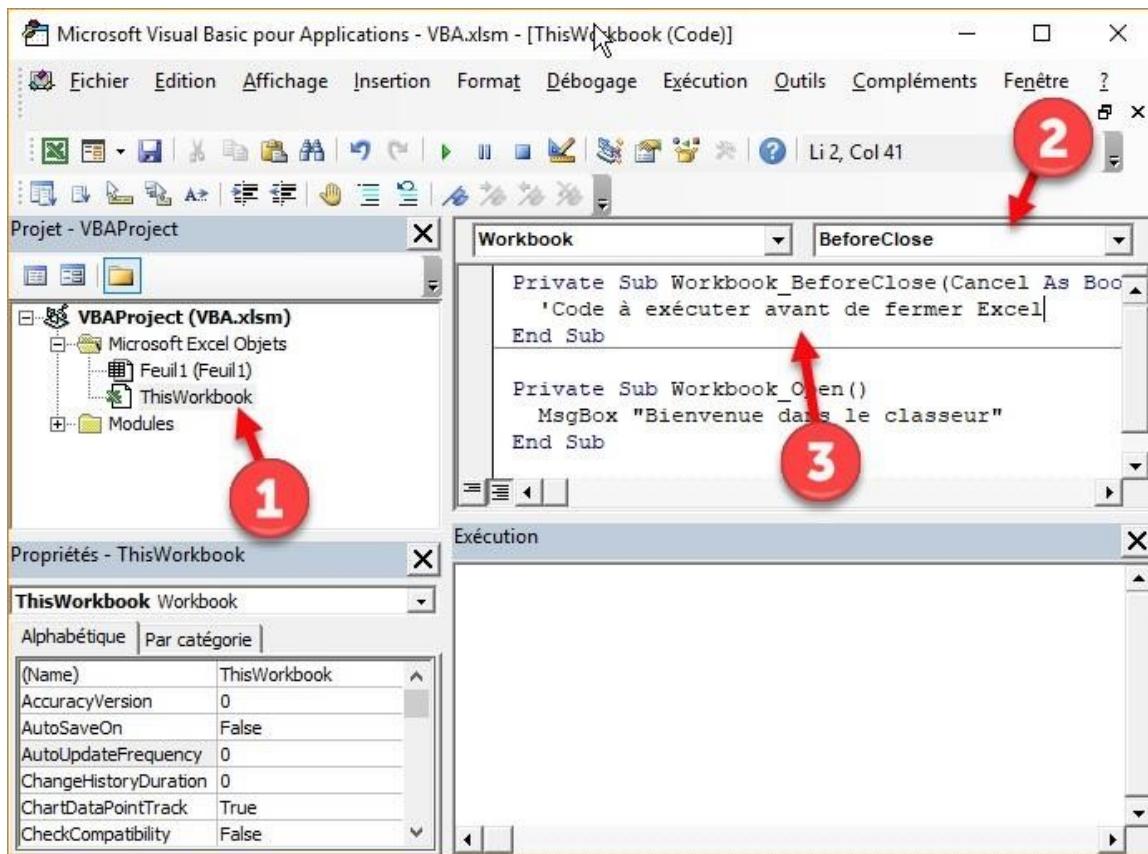
Lorsque vous ouvrirez le classeur, le code placé dans la procédure **Workbook_Open()** sera automatiquement exécuté :



Exécuter une procédure à la fermeture d'un classeur

Il peut parfois être utile d'exécuter une macro à la fermeture d'un classeur. Pour cela, vous devez créer la procédure **Workbook_BeforeClose()**.

Ouvrez la fenêtre Microsoft Visual Basic pour Applications du classeur concerné. Doublecliquez sur **ThisWorkbook** dans la fenêtre **Projet** (1) et sélectionnez **BeforeClose** dans la liste déroulante Procédure (2). La procédure **Workbook_BeforeClose()** est automatiquement créée. Il ne vous reste plus qu'à la compléter (3) :



Modifier la mise en forme des cellules

Couleur du texte

Dans les exemples de cette section, le texte devient rouge. Bien entendu, vous pouvez utiliser une autre couleur via la fonction **RGB()** ou en utilisant une couleur prédéfinie via **Font.ColorIndex**. Pour en savoir plus sur les couleurs utilisables en VBA, lisez l'article « Travailler avec des couleurs ».

Modifier la couleur du texte de la cellule active :

```
ActiveCell.Font.Color = RGB(255, 0, 0)
```

Modifier la couleur du texte de la sélection :

```
Selection.Font.Color = RGB(255, 0, 0)
```

Modifier la couleur du texte d'une plage de cellules sans la sélectionner (ici, la plage A3:C5) :

```
Range("A2:C5").Font.Color = RGB(255, 0, 0)
```

Modifier la couleur du texte de plusieurs lignes contigües (ici, les lignes 6 à 9) :

```
Rows("6:9").Font.Color = RGB(255, 0, 0)
```

Modifier la couleur du texte de plusieurs lignes disjointes (ici , les lignes 2, 5 et 9) :

```
Range("2:2, 5:5, 9:9").Font.Color = RGB(255, 0, 0)
```

Modifier la couleur du texte de plusieurs colonnes contigües (ici, les colonnes B à D) :

```
Columns("B:D").Font.Color = RGB(255, 0, 0)
```

Modifier la couleur du texte de plusieurs colonnes disjointes (ici , les colonnes A et C) :

```
Range("A:A, C:C").Font.Color = RGB(255, 0, 0)
```

Modifier la couleur du texte de toute la feuille courante :

```
Cells.Font.Color = RGB(255, 0, 0)
```

Modifier la couleur du texte d'une autre feuille de calcul (ici la feuille Feuil2) :

```
Feuil2.Cells.Font.Color = RGB(255, 0, 0) Couleur
```

d'arrière-plan

Tout ce qui a été dit dans la section précédente reste vrai ici, mais au lieu d'utiliser Font.Color, vous utiliserez Interior.Color ou Interior.ColorIndex.

Modifier la couleur de l'arrière-plan de la cellule active :

```
ActiveCell.Interior.Color = RGB(255, 0, 0)
```

Modifier la couleur de l'arrière-plan de la sélection :

```
Selection.Interior.Color = RGB(255, 0, 0)
```

Modifier la couleur de l'arrière-plan d'une plage de cellules sans la sélectionner (ici, la plage A3:C5) :

```
Range("A2:C5").Interior.Color = RGB(255, 0, 0)
```

Modifier la couleur de l'arrière-plan de plusieurs lignes contigües (ici , les lignes 6 à 9) :

```
Rows("6:9").Interior.Color = RGB(255, 0, 0)
```

Modifier la couleur de l'arrière-plan de plusieurs lignes disjointes (ici , les lignes 2, 5 et 9) :

```
Range("2:2, 5:5, 9:9").Interior.Color = RGB(255, 0, 0)
```

Modifier la couleur de l'arrière-plan de plusieurs colonnes contigües (ici, les colonnes B à D) :

```
Columns("B:D").Interior.Color = RGB(255, 0, 0)
```

Modifier la couleur de l'arrière-plan de plusieurs colonnes disjointes (ici , les colonnes A et C) :

```
Range("A:A, C:C").Interior.Color = RGB(255, 0, 0)
```

Modifier la couleur de l'arrière-plan de toute la feuille courante :

```
Cells.Interior.Color = RGB(255, 0, 0)
```

Modifier la couleur de l'arrière-plan d'une autre feuille de calcul (ici la feuille Feuil2) :

```
Feuil2.Cells.Interior.Color = RGB(255, 0, 0)
```

Police et attributs

Vous pouvez utiliser les propriétés **Name**, **Size**, **Bold**, **Italic**, **Underline** et **FontStyle** de l'objet **Font** pour modifier (respectivement) la police, la taille des caractères et les attributs gras, italique et souligné des cellules.

Par exemple, pour affecter la police Courier 16 points gras italique souligné à la plage A5:C3, vous pouvez écrire :

```
Range("A2:C5").Font.Name = "Courier"
```

```
Range("A2:C5").Font.Size = 16
```

```
Range("A2:C5").Font.Bold = True
```

```
Range("A2:C5").Font.Italic = True
```

```
Range("A2:C5").Font.Underline = True
```

Les attributs gras et italique peuvent être définis dans la propriété **FontStyle**. Cette propriété admet les valeurs suivantes :

- Regular
- Italic
- Bold
- Bold Italic

Le code devient :

```
Range("A2:C5").Font.Name = "Courier"
```

```
Range("A2:C5").Font.Size = 16
```

```
Range("A2:C5").Font.Underline = True
```

```
Range("A2:C5").Font.FontStyle = "Bold Italic"
```

Il est encore possible d'améliorer le code. En factorisant Range(« A2:C5 »).Font, le code devient :

```
Sub test()  
  
    With Range("A2:C5").Font  
  
        .Name = "Courier"  
  
        .Size = 16  
  
        .Underline = True  
  
        .FontStyle = "Bold Italic"  
  
    End With
```

Format des cellules

Vous pouvez utiliser des instructions VBA pour définir le format des cellules : nombre, monétaire, date, etc. Pour cela, vous utiliserez la propriété **NumberFormat** d'un objet **Range**, **Rows**, **Columns**, **Cells** ou **ActiveCell**.

Format de date et d'heure

Pour définir un format date ou heure, vous utiliserez les codes de format suivants :

Codes de format Signification

yyyy	Année sur 4 chiffres
yy	Année sur 2 chiffres
mmmm	Mois en texte
mmm	Mois en texte abrégé
mm	Mois entre 01 et 12
m	Mois entre 1 et 12
dddd	Jour en texte
dd	Jour en texte abrégé
dd	Jour entre 01 et 31
d	Jour entre 1 et 31
hh	Heures avec zéro entre 00 et 23
h	Heures sans zéro entre 0 et 23
nn	Minutes avec zéro entre 00 et 59
n	Minutes sans zéro entre 0 et 59

ss Secondes avec zéro entre 00 et 59 s Secondes sans zéro entre 0 et 59

Par exemple, pour que la cellule courante affiche une date au format samedi 12/08/2018, vous utiliserez cette instruction :

```
ActiveCell.NumberFormat = "dddd dd/mm/yyyy"
```

Voici un exemple de rendu :

	A	B
12		
13		samedi 12/08/2017
14		
15		
16		
17		

Ou encore, pour que la cellule courante affiche une information horaire avec un séparateur « : » et comprenant les heures sur 2 chiffres, les minutes sur 2 chiffres et les secondes sur 2 chiffres, vous utiliserez cette instruction :

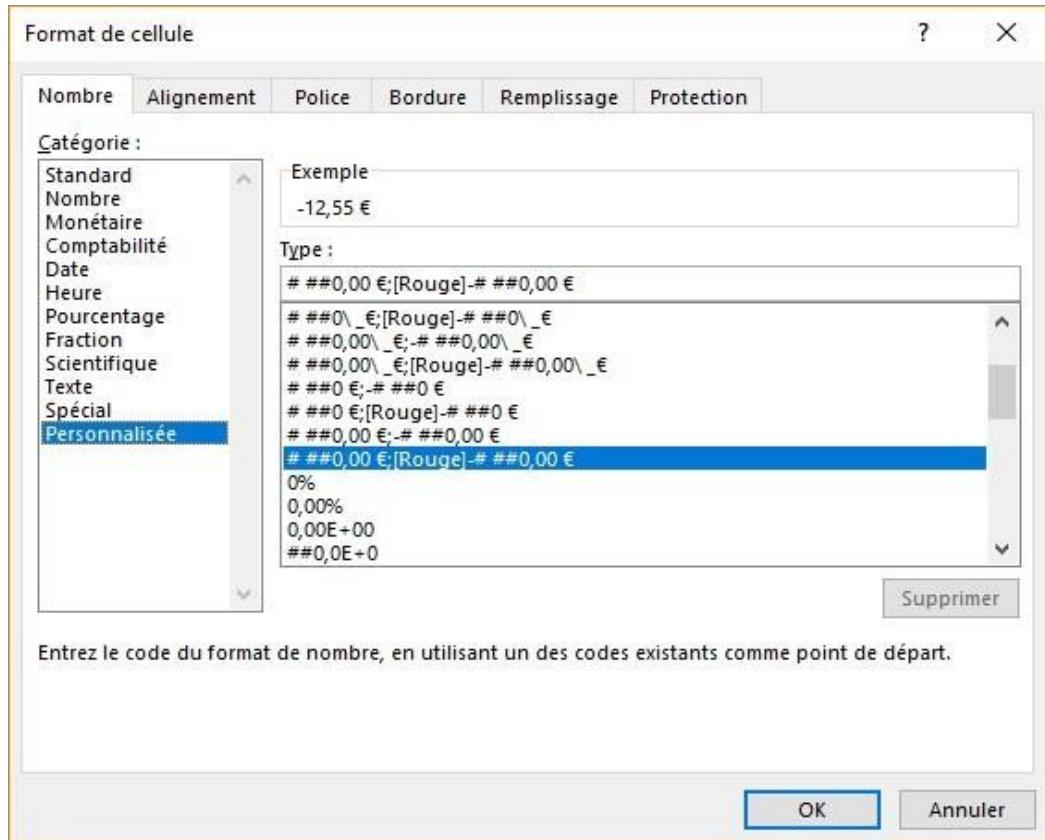
```
ActiveCell.NumberFormat = "hh:mm:ss"
```

Voici un exemple de rendu :

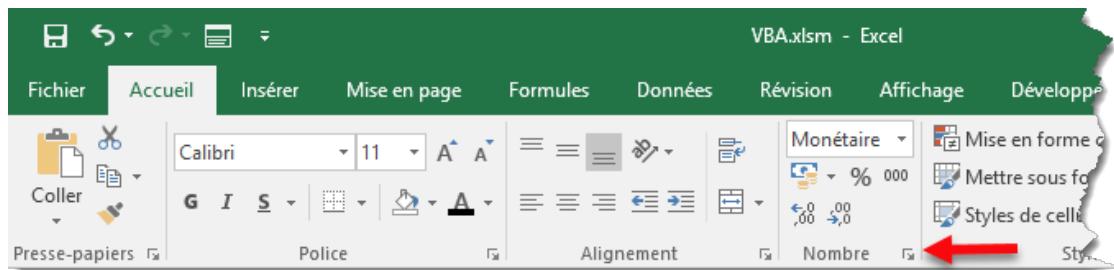
	A	B	
12			
13		12:08:15	
14			
15			
16			

Format numérique

Pour faciliter la saisie du format des cellules numériques, je vous conseille d'afficher la boîte de dialogue **Format de cellule** et de sélectionner la catégorie **Personnalisée** sous l'onglet **Nombre** :



Pour afficher cette boîte de dialogue, cliquez sur l'icône **Format de nombre**, dans le groupe **Nombre**, sous l'onglet **Accueil** du ruban :



Les chaînes de formatage de la boîte de dialogue Format de cellule seront appliqués aux cellules avec la propriété NumberFormatLocal. Par exemple, pour appliquer à la cellule active un format monétaire avec un affichage en rouge pour les valeurs négatives, vous utiliserez cette instruction :

```
ActiveCell.NumberFormatLocal = "# ##0,00 €;[Rouge]-# ##0,00 €"
```

Voici un exemple de rendu :

	A	B
12		
13		-12,55 €
14		
15		
16		

Copier-coller en VBA

Une seule ligne de code suffit pour copier-coller des cellules, plages, lignes ou colonnes. Voici quelques exemples :

Copie de la cellule G5 en G30 :

```
Range("G5").Copy Range("G30")
```

Copie de la colonne A dans la colonne G :

```
Range("A:A").Copy Range("G:G")
```

Copie de la ligne 13 dans la ligne 25 :

```
Range("13:13").Copy Range("25:25")
```

Copie de la plage A2:C3 de la feuille courante à partir de la cellule A1 du classeur Feuil2 :

```
Range("A2:C3").Copy Worksheets("Feuil2").Range("A1") Remarque
```

Ces instructions n'utilisent pas le presse-papiers multiple d'Office.

Dans tous ces exemples, le contenu des cellules et leur mise en forme sont copies-collés. Si vous voulez limiter le copier-coller aux valeurs ou à la mise en forme, vous devrez utiliser un collage spécial. Ici par exemple, les valeurs de la plage **A2:B8** sont copiées en **C12**, sans tenir compte de la mise en forme :

```
Set range1 = Range("A2:B8")
range1.Copy
Range("C12").Select
ActiveCell.PasteSpecial Paste:=xlPasteValues
```

Vous voulez copier-coller la mise en forme mais pas la valeur des cellules ? Remplacez la constante **xlPasteValues** par la constante **xlPasteFormats**. Le code devient :

```
Set range1 = Range("A2:B8")
range1.Copy
Range("C12").Select
ActiveCell.PasteSpecial Paste:=xlPasteFormats
```

Utiliser les fonctions d'Excel en VBA

De très nombreuses fonctions sont accessibles dans Excel. Ces fonctions peuvent parfois rendre de grands services en VBA et éviter la saisie de nombreuses instructions. Les fonctions d'Excel sont accessibles via l'objet **WorksheetFunction**.

Voici quelques exemples d'utilisation.

Min, Max et Average

Supposons que vous recherchez les valeurs minimales et maximales d'une plage de cellules et que vous vouliez calculer la moyenne des valeurs de la plage. Pour cela, vous pouvez utiliser les fonctions **Min()**, **Max()** et **Average()** d'Excel.

Nous allons partir de cette feuille de calcul. La plage examinée sera **A1:B5** :

	A	B	C
1	10	16	
2	50	22	
3	25	18	
4	12	35	
5	14	40	
6			
7			

Voici le code utilisé :

```
Dim minimum, maximum As Integer  
  
Dim moyenne As Single  
  
Set plage = Worksheets("Feuill1").Range("A1:B5")  
minimum =  
Application.WorksheetFunction.min(plage)  
maximum =  
Application.WorksheetFunction.max(plage)  
moyenne =  
WorksheetFunction.Average(plage)  
  
MsgBox "Valeur minimale : " & minimum  
  
MsgBox "Valeur maximale : " & maximum  
  
MsgBox "Moyenne : " & moyenne
```

Et voici le résultat :

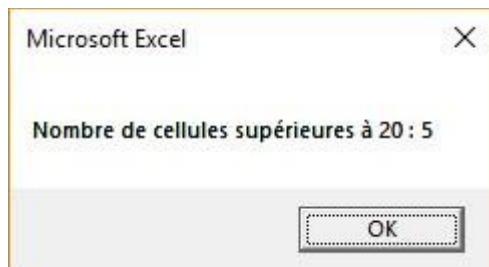


Nombre de cellules supérieures à 20

Toujours à partir de la même feuille de calcul, calcul du nombre de cellules de la plage A1:B5 dont la valeur est supérieure à 20 :

```
Dim grand As Integer
grand = WorksheetFunction.CountIf(Range("A1:B5"), ">20")
MsgBox "Nombre de cellules supérieures à 20 : " & grand
```

Voici le résultat :

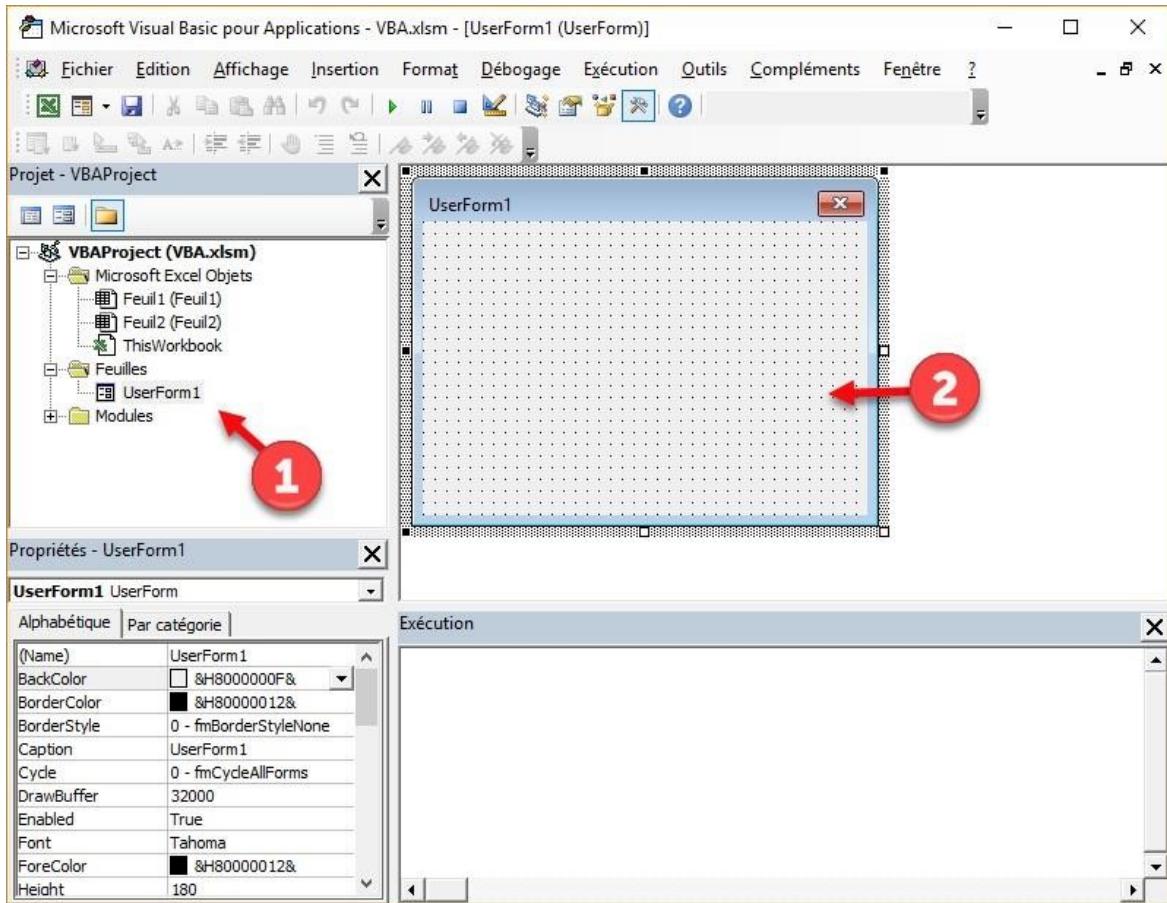


UserForms

Les UserForms sont des boîtes de dialogue personnalisées. Elles reposent sur l'utilisation de l'objet UserForm. Cette section va vous montrer comment les utiliser pour créer des interfaces utilisateur adaptées à vos projets pour afficher mais aussi saisir des données.

Définition d'une UserForm

Pour définir une UserForm dans le projet en cours, rendez-vous dans la fenêtre Microsoft Visual Basic pour Applications et lancez la commande **UserForm** dans le menu **Insertion**. Un objet **UserForm** est alors ajouté dans la fenêtre **Projet**, sous **Feuilles** (1) et la boîte de dialogue personnalisée apparaît dans l'interface (2) :

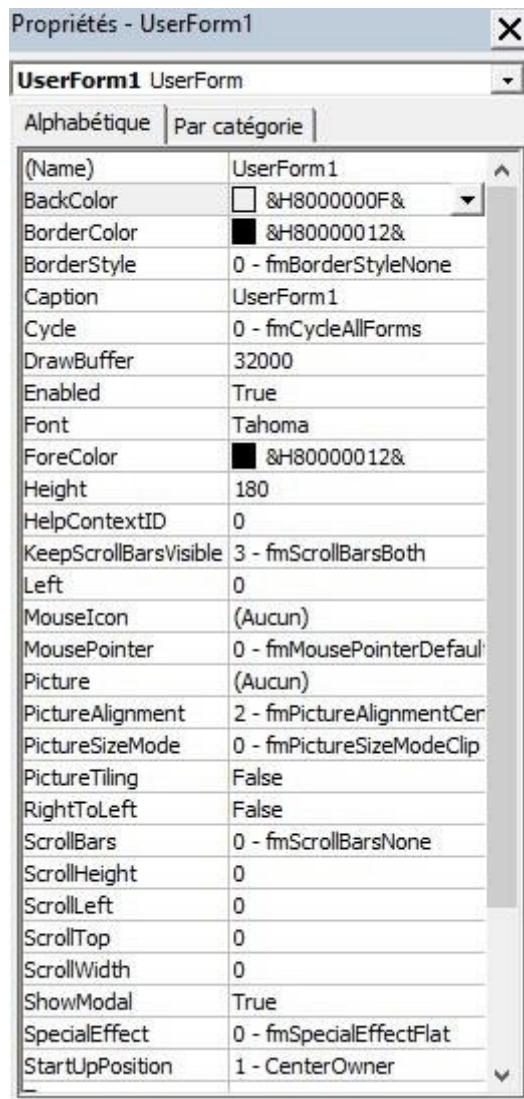


Personnalisation de la boîte de dialogue personnalisée

Vous pouvez librement redimensionner votre boîte de dialogue personnalisée en agissant sur ses poignées de redimensionnement. Pour l'afficher en mode exécution, appuyez simplement sur la touche de fonction **F5** ou lancez la commande **Exécuter Sub/UserForm** dans le menu **Exécution** :



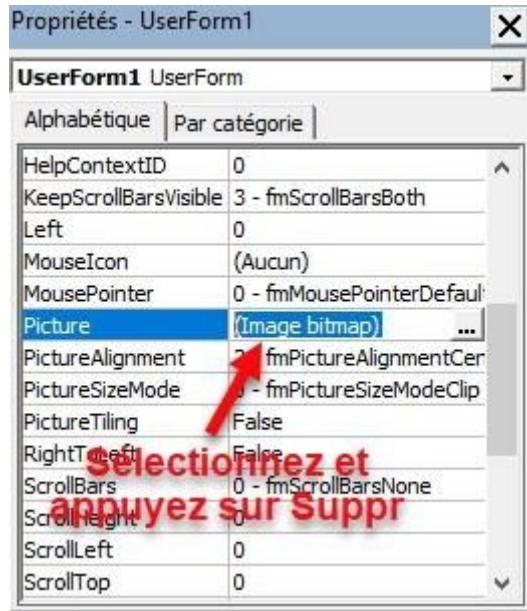
Les propriétés de la boîte de dialogue personnalisée sont regroupées dans la fenêtre **Propriétés**, dans l'angle inférieur gauche de la fenêtre Microsoft Visual Basic pour Applications. Il suffit de les modifier pour changer l'allure de votre UserForm :



Toutes ces propriétés sont faciles à comprendre. Personnellement, j'utilise essentiellement les propriétés suivantes :

Propriété	Signification
Top et Left	Position de la boîte de dialogue personnalisée sur l'écran
Caption	Titre de la boîte de dialogue personnalisée
Font	Texte utilisé par défaut dans les contrôles
BackColor	Couleur d'arrière-plan de la boîte de dialogue personnalisée
Picture	Image d'arrière-plan de la boîte de dialogue personnalisée

Si vous insérez une image d'arrière-plan dans un UserForm et que vous changez d'avis, vous vous demandez peut-être comment la supprimer. Eh bien, il suffit de sélectionner la valeur de la propriété **Picture** dans la fenêtre **Propriétés** et d'appuyer sur la touche *Suppr* du clavier :



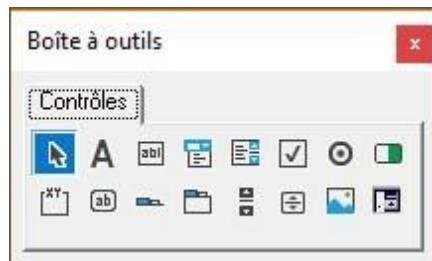
Vous pourriez également choisir de supprimer l'image d'arrière-plan à l'exécution de la boîte de dialogue personnalisée (même si la suppression de l'image dans les propriétés de votre UserForm semble plus « propre »). Pour cela, insérez la commande suivante dans la procédure **UserForm_Activate()** :

```
UserForm1.Picture = LoadPicture()
```

Insertion de contrôles dans une boîte de dialogue personnalisée

Dans les sections à venir, nous allons passer en revue les différents contrôles utilisables dans une boîte de dialogue personnalisée et montrer comment y accéder à l'aide d'instructions VBA.

Avant tout, commencez par afficher la boîte à outils avec la commande **Boîte à outils** dans le menu **Affichage** :



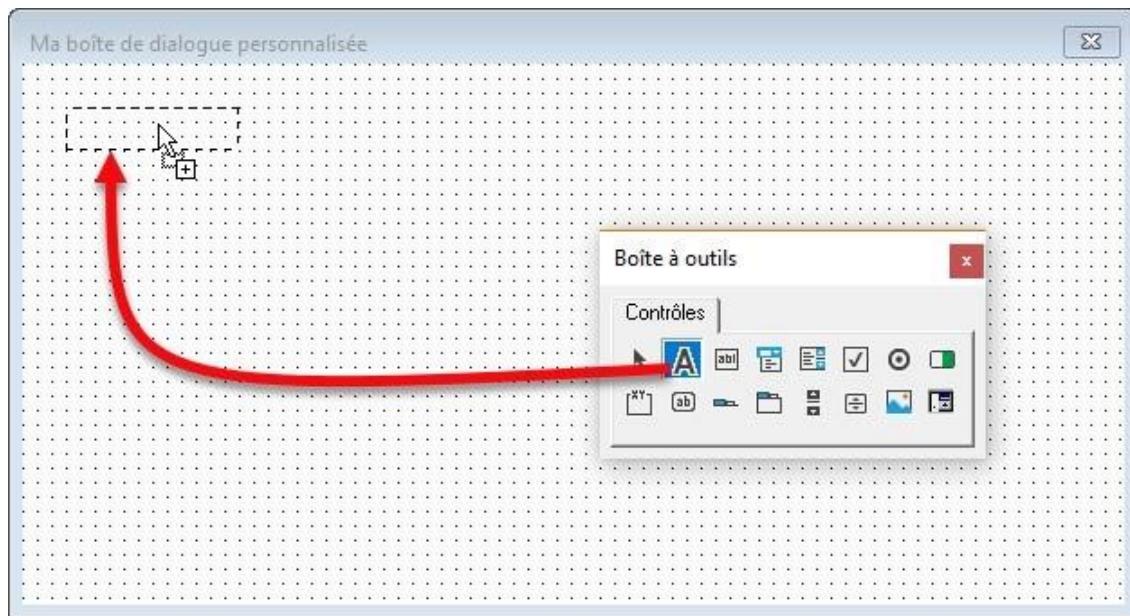
Remarque

Si la commande **Boîte à outils** est grisée dans le menu **Affichage**, c'est certainement parce que vous êtes en mode d'affichage **Code** et non **Objet**. Appuyez simultanément sur les touches *Maj* et *F7* ou lancez la commande **Objet** dans le menu **Affichage**. La commande **Boîte à outils** devrait maintenant être accessible dans le menu **Affichage**.

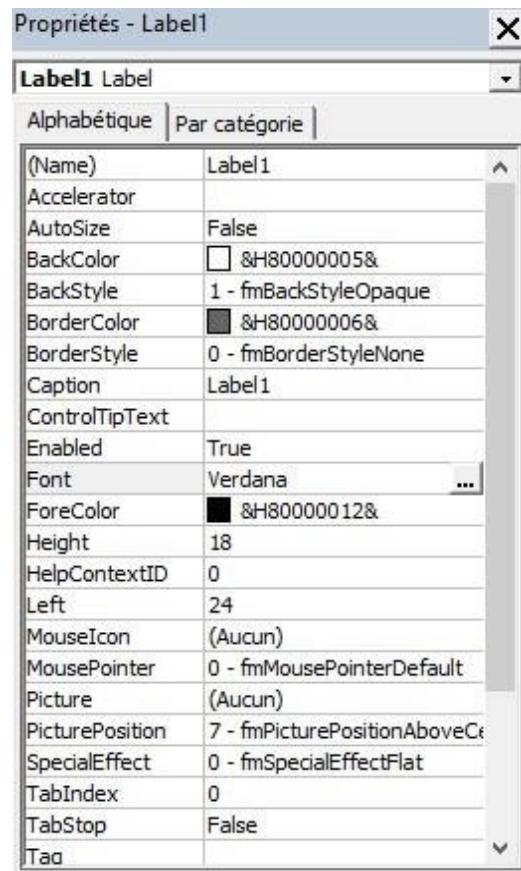
Labels dans une boîte de dialogue personnalisée

Vous utiliserez des labels pour afficher des informations textuelles, comme par exemple des étiquettes devant des zones de texte, des cases à cocher ou des boutons radio.

Pour insérer un Label, glissez-déposez un contrôle **Label** sur la boîte de dialogue personnalisée :



Au relâchement du bouton gauche de la souris, le **Label** est inséré dans la boîte de dialogue personnalisée et ses propriétés sont accessibles dans la fenêtre **Propriétés**. Si nécessaire, utilisez cette fenêtre pour modifier l'allure du Label. Vous utiliserez certainement les propriétés **Caption**, **Font** et **ForeColor** :



Si nécessaire, vous pouvez lire ou modifier les propriétés de l'objet **Label** une fois la boîte de dialogue personnalisée affichée. Par exemple, pour modifier le texte du label, vous pourriez écrire :

```
Label1.Caption = "Nouveau texte dans le Label"
```

Vous pouvez également mettre en place des fonctions événementielles sur un Label. Si vous êtes en affichage **Objet**, appuyez sur *F7* ou lancez la commande **Code** dans le menu **Affichage** pour passer en affichage **Code**. Sélectionnez **Label1** dans la première liste déroulante et un **événement** dans la seconde. La procédure événementielle est immédiatement créée. A titre d'exemple, le code suivant modifie la couleur du premier label lorsque l'utilisateur clique dessus. Si le texte est noir, il devient rouge. Dans le cas contraire, il devient noir :

```
Private Sub Label1_Click()  
  
If Label1.ForeColor = RGB(0, 0, 0) Then  
  
    Label1.ForeColor = RGB(255, 0, 0)  
  
Else  
  
    Label1.ForeColor = RGB(0, 0, 0)  
  
End If  
  
End Sub
```

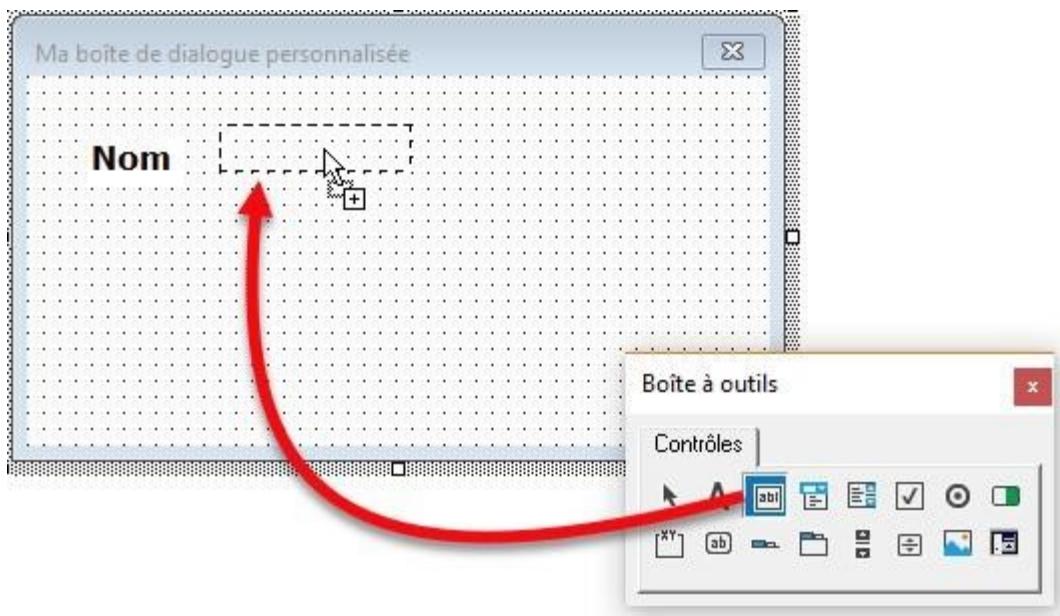
En sélectionnant **Label1** dans la première liste et **Activate** dans la seconde, vous pouvez choisir la couleur du label à l'ouverture de la boîte de dialogue personnalisée. Ici, noir :

```
Private Sub UserForm_Activate()  
  
    Label1.ForeColor = RGB(0, 0, 0)  
  
End Sub
```

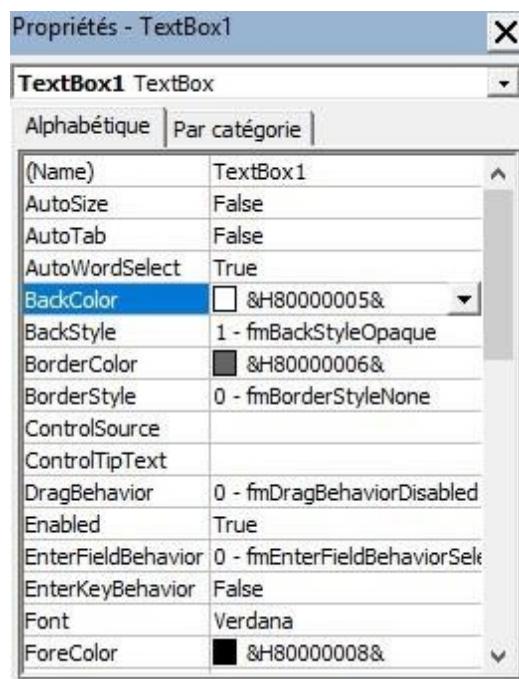
TextBox dans une boîte de dialogue personnalisée

Vous utiliserez des TextBox pour permettre à l'utilisateur de saisir des informations alphanumériques.

Pour insérer un TextBox, glissez-déposez un contrôle TextBox sur la boîte de dialogue personnalisée :



Tout comme pour un Label, vous pouvez utiliser la fenêtre des propriétés pour modifier les caractéristiques de l'objet TextBox que vous venez d'insérer :



Vous pouvez également utiliser des instructions VBA pour accéder en lecture ou en écriture aux propriétés de l'objet TextBox. Par exemple, pour recopier dans la cellule A1 de la feuille courante la valeur saisie dans la zone de texte TextBox1 à chaque frappe au clavier, vous définirez la procédure évènementielle TextBox1_KeyUp() suivante :

```
Private Sub TextBox1_KeyUp(ByVal KeyCode As MSForms.ReturnInteger, ByVal Shift As Integer)

    Cells(1, 1).Value = TextBox1.Text

End Sub
```

Des ComboBox dans une boîte de dialogue personnalisée

Vous utiliserez des contrôles ComboBox pour insérer des listes déroulantes dans une boîte de dialogue personnalisée.

Pour insérer un ComboBox, glissez-déposez un contrôle **ComboBox** sur la boîte de dialogue personnalisée :

La ComboBox peut être remplie en faisant référence à un bloc de cellules d'une feuille de calcul ou à l'aide de code VBA.

Remplissage avec un bloc de cellules

Pour référencer un bloc de cellules, renseignez la propriété **RowSource** du contrôle **ComboBox** :

En supposant que les cellules A2 à A6 de la feuille Feuil2 contiennent les données suivantes :

Voici le contenu du ComboBox :

Vous voulez affecter une valeur par défaut à la liste déroulante ? Rien de plus simple : affectez l'index de cette valeur à la propriété **ListIndex** du contrôle (la valeur 0 correspond au premier élément de la liste). Cette affectation se fera par exemple dans la procédure **UserForm_Activate()**. Par exemple, pour choisir Nathalie par défaut (la troisième valeur de la liste), vous utiliserez cette instruction dans la procédure **UserForm_Activate()** :

```
Private Sub UserForm_Activate()
```

```
    ComboBox1.ListIndex = 2
```

```
End Sub
```

Remplissage avec du code VBA

Pour remplir un contrôle ComboBox avec du code VBA, vous utiliserez la méthode **AddItem**. Ici par exemple, on insère les prénoms **Bertrand**, **Pierre**, **Nathalie**, **Pierric** et **Liliane** dans le contrôle **ComboBox1** :

```

Private Sub UserForm_Initialize()

    ComboBox1.AddItem "Bertrand"

    ComboBox1.AddItem "Pierre"

    ComboBox1.AddItem "Nathalie"

    ComboBox1.AddItem "Pierric"

    ComboBox1.AddItem "Liliane"

End Sub

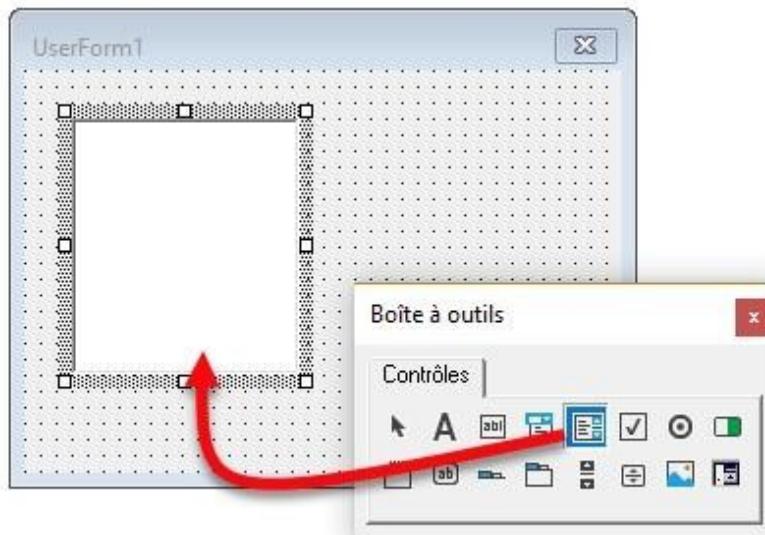
```

Le résultat est le même que précédemment :

Des ListBox dans une boîte de dialogue personnalisée

Vous utiliserez des contrôles **ListBox** pour insérer des zones de liste dans une boîte de dialogue personnalisée. Ces contrôles permettent de choisir un ou plusieurs éléments dans une liste de choix.

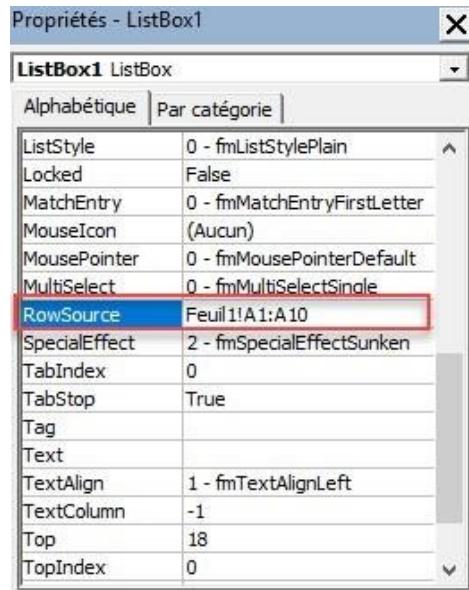
Pour insérer un ListBox, glissez-déposez un contrôle **ListBox** sur la boîte de dialogue personnalisée :



Le ListBox peut être rempli en faisant référence à un bloc de cellules d'une feuille de calcul ou à l'aide de code VBA.

Remplissage avec un bloc de cellules

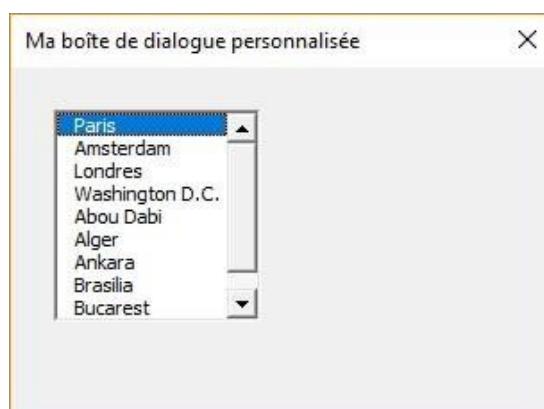
Vous pouvez remplir un ListBox avec un bloc de cellules dans une feuille de calcul. Pour cela, renseignez la propriété **RowSource** du contrôle **ListBox** :



En supposant que les cellules **A1** à **A10** de la feuille **Feuil1** contiennent les données suivantes :

	A	B
1	Paris	
2	Amsterdam	
3	Londres	
4	Washington D.C.	
5	Abou Dabi	
6	Alger	
7	Ankara	
8	Brasilia	
9	Bucarest	
10	Varsovie	
11		
12		

Voici le contenu du ListBox :



Vous voulez affecter une valeur par défaut à la zone de liste ? Rien de plus simple : affectez l'index de cette valeur à la propriété **ListIndex** du contrôle (la valeur **0** correspond au premier élément de la liste). Cette affectation se fera par exemple dans la procédure **UserForm_Activate()**. Par exemple, pour choisir **Abou Dabi** par défaut (la cinquième valeur de la liste), vous utiliserez cette instruction dans la procédure **UserForm_Activate()** :

```
Private Sub UserForm_Activate()  
  
    ListBox1.ListIndex = 4  
  
End Sub
```

Remplissage avec du code VBA

Pour remplir un contrôle ListBox avec du code VBA, vous utiliserez la méthode **AddItem**. Ici par exemple, on insère les capitales **Paris, Amsterdam, Londres, Washington D.C., Abou Dabi, Alger, Ankara, Brasilia, Bucarest et Varsovie** dans le contrôle **ListBox1** :

```
Private Sub UserForm_Activate()  
  
    ListBox1.AddItem "Paris"  
  
    ListBox1.AddItem "Amsterdam"  
  
    ListBox1.AddItem "Londres"  
  
    ListBox1.AddItem "Washington D.C."  
  
    ListBox1.AddItem "Abou Dabi"  
  
    ListBox1.AddItem "Alger"  
  
    ListBox1.AddItem "Ankara"  
  
    ListBox1.AddItem "Brasilia"  
  
    ListBox1.AddItem "Bucarest"  
  
    ListBox1.AddItem "Varsovie"  
  
End Sub
```

Le résultat est le même que précédemment :



Des CheckBox dans une boîte de dialogue personnalisée

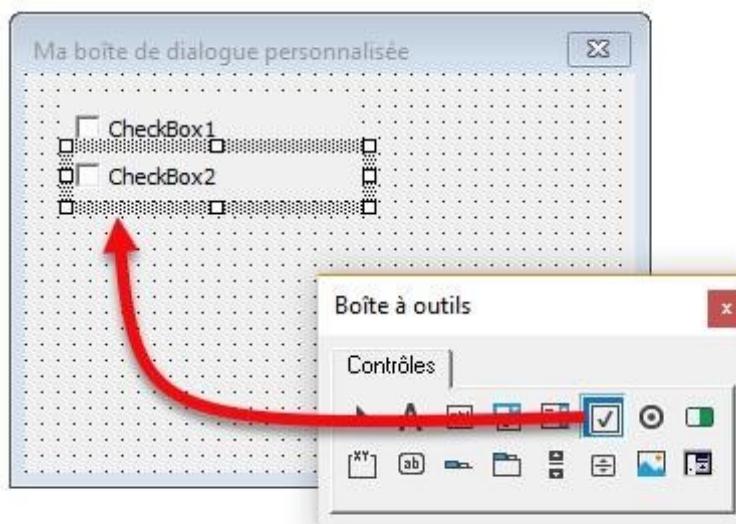
8 décembre 2017

Les **CheckBox** (cases à cocher) sont utilisées lorsque l'utilisateur peut sélectionner une ou plusieurs options dans une liste d'options.

La propriété **Value** du CheckBox vaut :

- **True** lorsque le CheckBox est coché.
- **False** lorsque le CheckBox est décoché.

Pour insérer un CheckBox, glissez-déposez un contrôle **CheckBox** sur la boîte de dialogue personnalisée :



La procédure **CheckBox_Change()** peut être utilisée pour réagir aux changements d'état d'un Checkbox. Supposons par exemple que le CheckBox **CheckBox1** ait été inséré dans une boîte

de dialogue personnalisée. Vous pourriez écrire la procédure **CheckBox1_Change()** pour modifier l'étiquette du CheckBox en fonction de son état (coché ou décoché) :

```
Private Sub CheckBox1_Change()  
  
    If CheckBox1.Value = True Then  
  
        CheckBox1.Caption = "Case cochée"  
  
    Else  
  
        CheckBox1.Caption = "Case décochée"  
  
    End If  
  
End Sub
```

Vous pourriez également afficher le texte **Case décochée** dans l'étiquette du CheckBox à l'ouverture de la boîte de dialogue personnalisée *via* la procédure **UserFormActivate()** :

```
Private Sub UserForm_Activate()  
  
    CheckBox1.Caption = "Case décochée"  
  
End Sub
```

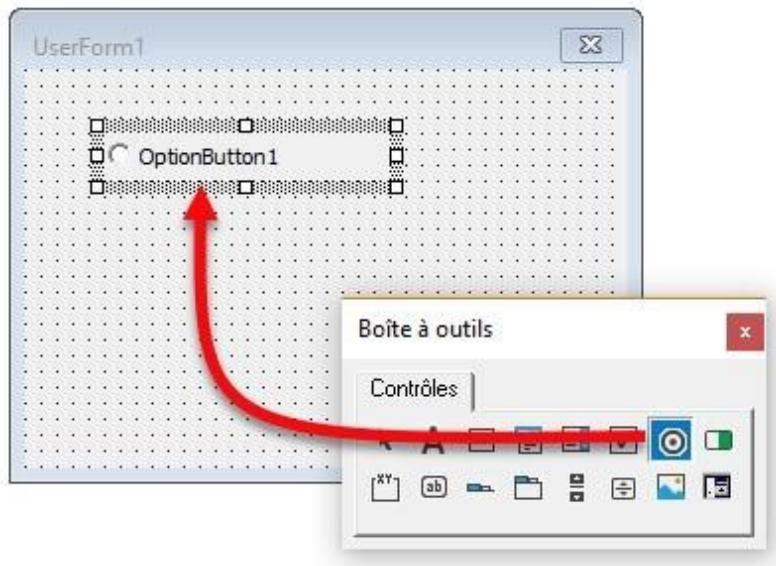
Des OptionButton dans une boîte de dialogue personnalisée

Les **OptionButton** (boutons radio) sont généralement utilisés lorsque l'utilisateur peut effectuer un choix et un seul parmi plusieurs. Ils sont alors placés dans un cadre (**Frame**) pour améliorer la présentation.

La propriété **Value** du OptionButton vaut :

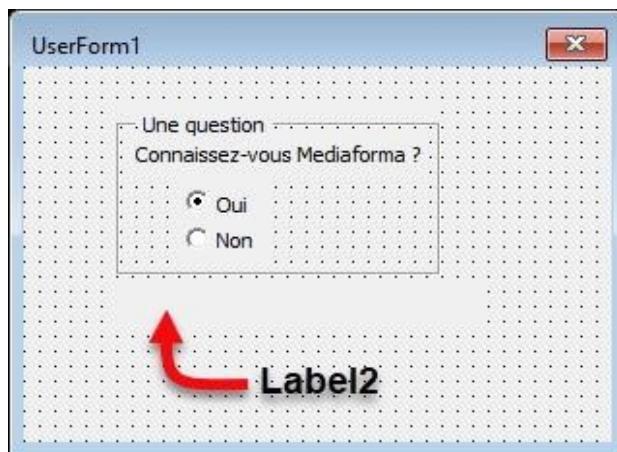
- **True** lorsque l'OptionButton est sélectionné.
- **False** lorsque l'OptionButton est désélectionné.

Pour insérer un OptionButton, glissez-déposez un contrôle **OptionButton** sur la boîte de dialogue personnalisée :



Pour relier deux ou plusieurs OptionButton de telle sorte qu'un seul d'entre eux soit sélectionné à la fois, affectez la même valeur à la propriété **GroupName** de ces contrôles.

La procédure **OptionButton_Change()** peut être utilisée pour réagir aux changements d'état d'un OptionButton. Supposons par exemple que deux OptionButton de même GroupName aient été insérés dans un contrôle Frame.



Pour qu'un message s'affiche dans le contrôle **Label2** lorsque l'utilisateur sélectionne **Oui** ou **Non**, vous utiliserez ce code :

```

Private Sub OptionButton1_Change()
    Label2.Caption = "Parfait !"
End Sub

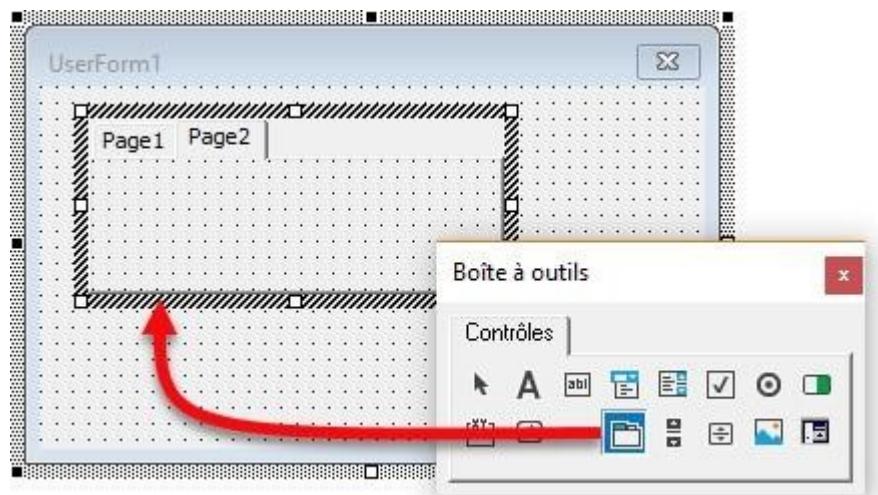
Private Sub OptionButton2_Change()
    Label2.Caption = "Cela ne saurait tarder"
End Sub

```

Le contrôle MultiPage dans une boîte de dialogue personnalisée

Les contrôles **MultiPage** sont intéressants lorsque de nombreux contrôles doivent être placés dans une boîte de dialogue personnalisée. Ils donnent accès à plusieurs pages *via* des onglets. Par défaut deux onglets sont créés, mais il est très simple d'en ajouter en mode conception ou avec du code VBA.

Pour insérer un contrôle MultiPage, glissez-déposez un contrôle **MultiPage** sur la boîte de dialogue personnalisée :

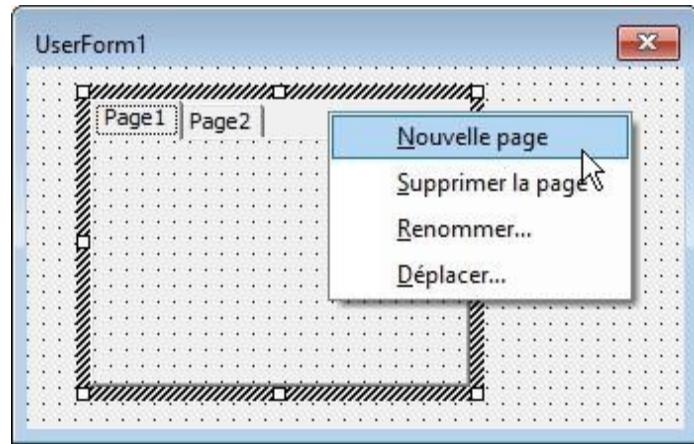


Lorsque vous sélectionnez une page en sélectionnant sur son onglet, plusieurs propriétés intéressantes peuvent être utilisées :

Propriété	Signification
Caption	Texte dans l'onglet
Enabled	Accessible (True) ou non (False)
Picture	Arrière-plan de la page
Visible	Onglet de la page affiché (True) ou caché (False)

La propriété **Value** du contrôle **MultiPage** détermine quelle page est affichée : **0** représente la première page, **1** représente la deuxième, ainsi de suite...

Pour ajouter une page en mode conception, cliquez du bouton droit dans la barre des onglets et sélectionnez **Nouvelle page** dans le menu :



Pour faire la même chose en VBA, vous utiliserez ces instructions :

```
Set p3 = MultiPage1.Pages.Add
```

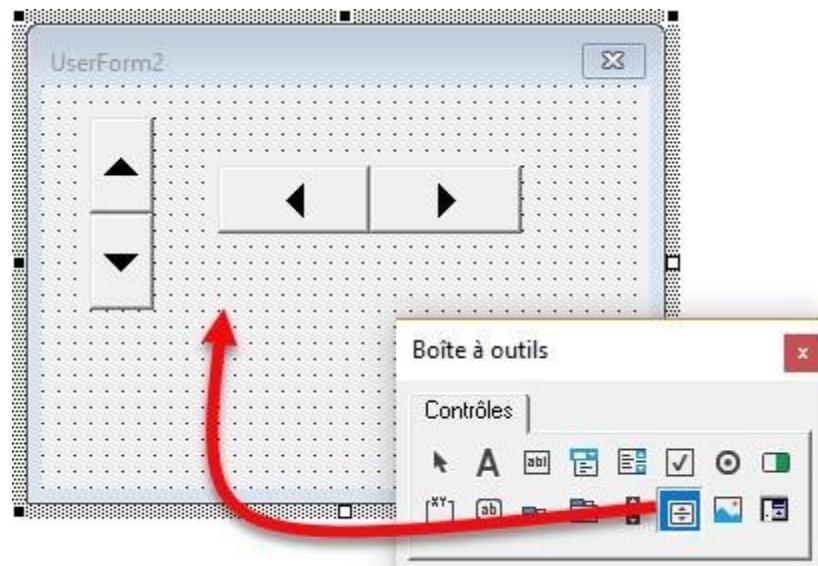
```
p3.Caption = "Page 3"
```

Une fois le contrôle MultiPage en place, il ne vous reste plus qu'à insérer les contrôles voulu dans les différentes pages.

Des SpinButton dans une boîte de dialogue personnalisée

Le contrôle **SpinBox** (toupie) permet d'incrémenter ou de décrémenter une valeur d'un certain pas entre une valeur minimale et une valeur maximale.

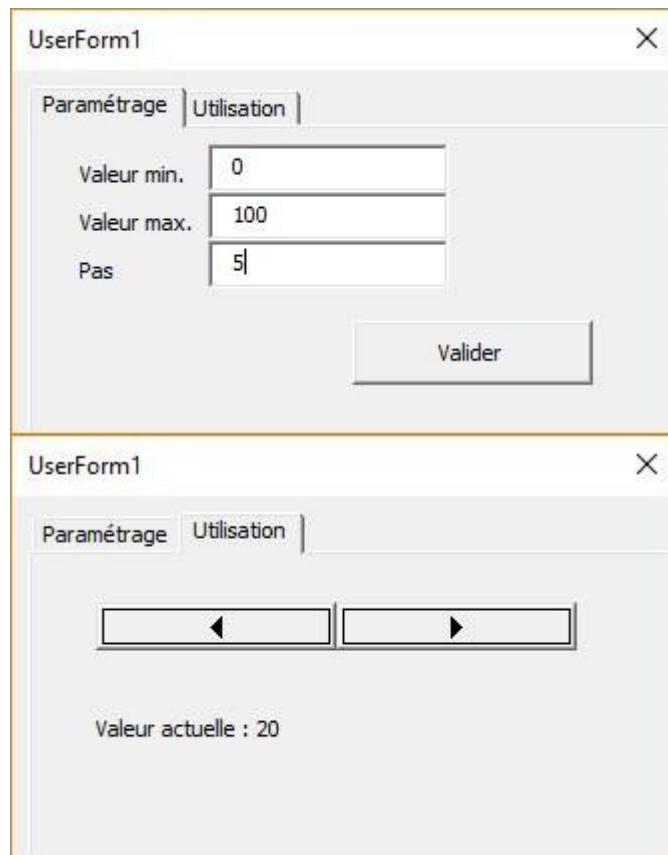
Pour insérer un contrôle SpinButton, sélectionnez l'outil **Toupie** dans la boîte à outils, puis tracez le contour voulu dans la boîte de dialogue personnalisée. Selon votre tracé, le contrôle sera horizontal ou vertical :



Vous utiliserez essentiellement les propriétés suivantes dans un SpinButton :

Propriété	Signification
Min	Valeur minimale
Max	Valeur maximale
SmallChange	Pas de progression

A titre d'exemple, nous allons définir une boîte de dialogue personnalisée qui contient un contrôle **MultiPage** composé de deux pages : une pour définir les caractéristiques du **SpinButton** et une autre pour utiliser le **SpinButton** :



La mise en place des contrôles n'offre aucune difficulté.

Pour faire fonctionner cette boîte de dialogue personnalisée, nous allons écrire quelques lignes de code VBA.

Au clic sur le bouton de la première page :

- Les valeurs saisies dans les trois zones de texte sont affectées aux propriétés **Min**, **Max** et **SmallChange** du SpinButton. □ Le texte “Valeur actuelle : 0” est affiché dans le label de la deuxième page.
- La boîte de dialogue bascule automatiquement sur la deuxième page.

```
Private Sub CommandButton1_Click()
```

```
    With SpinButton1
```

```

    .Min = TextBox1.Value
    .Max = TextBox2.Value
    .SmallChange = TextBox3.Value
End With

Label4.Caption = "Valeur actuelle : 0"

MultiPage1.Value = 1 'Active la deuxième page
End Sub

```

Pour compléter la prise en compte du clic sur le bouton, il reste à prendre en compte le clic sur les boutons du SpinButton. La valeur du SpinButton est alors automatiquement modifiée. Il suffit de l'afficher dans le label :

```

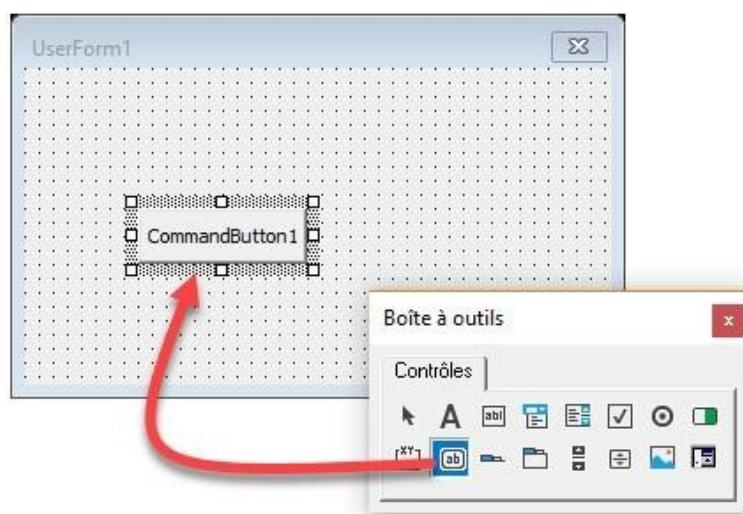
Private Sub SpinButton1_Change()
    Label4.Caption = "Valeur actuelle : " & SpinButton1.Value
End Sub

```

Des CommandButton dans une boîte de dialogue personnalisée

Le contrôle **CommandButton** (bouton) permet essentiellement d'exécuter du code lorsque l'utilisateur clique dessus.

Pour insérer un contrôle **CommandButton**, sélectionnez l'outil **CommandButton** dans la boîte à outils, puis cliquez dans la boîte de dialogue personnalisée :



Vous utiliserez :

- La propriété **Caption** pour définir le texte qui apparaît sur le bouton.
- L'événement **Click** pour réagir au clic sur le bouton.

A titre d'exemple, nous allons définir une boîte de dialogue personnalisée qui contient un **CommandButton**. Lorsque l'utilisateur clique sur ce bouton, un message sera affiché à l'aide de la méthode **MsgBox**.

Voici la boîte de dialogue personnalisée :



Un **CommandButton** est inséré, puis sa propriété **Caption** est initialisée à "Cliquez ici".

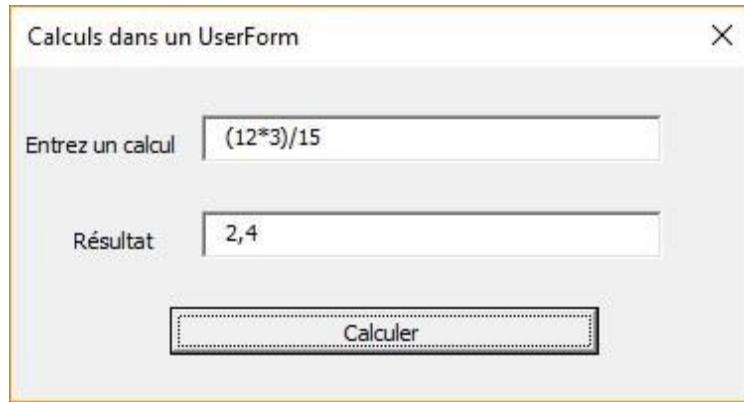
Double-cliquez sur le bouton et complétez la procédure **CommandButton1_Click()** comme ceci :

```
Private Sub CommandButton1_Click()  
  
    MsgBox "Merci d'avoir cliqué"  
  
End Sub
```

Il ne reste plus qu'à lancer la boîte de dialogue personnalisée en cliquant sur le bouton **Exécuter**, ou en appuyant sur la touche de fonction *F5* :

Des calculs dans un UserForm

Dans cette section, je vais vous montrer comment exécuter un calcul mathématique au clic sur un bouton et afficher le résultat dans un **TextBox**. Voici le résultat attendu :



Créez une nouvelle boîte de dialogue personnalisée. Affectez la valeur “Calculs dans un UserForm” à sa propriété **Caption**.

Aoutez deux **Label**, deux **TextBox** et un **CommandButton**.

Affectez la valeur :

- “Entrez un calcul” à la propriété **Caption** du premier **Label**.
- “Résultat” à la propriété **Caption** du deuxième **Label**.
- “Calculer” à la propriété **Caption** du **CommandButton**.

Pour obtenir le résultat du calcul entré dans le premier **TextBox**, nous allons utiliser une fonction très pratique : **Evaluate()**. Cette fonction demande un argument de type **String** qui contient un calcul. Elle retourne le résultat du calcul.

Double-cliquez sur le bouton. L’affichage bascule sur la fenêtre **Code** et la procédure **CommandButton1_Click()** est créée. Complétez-la comme ceci :

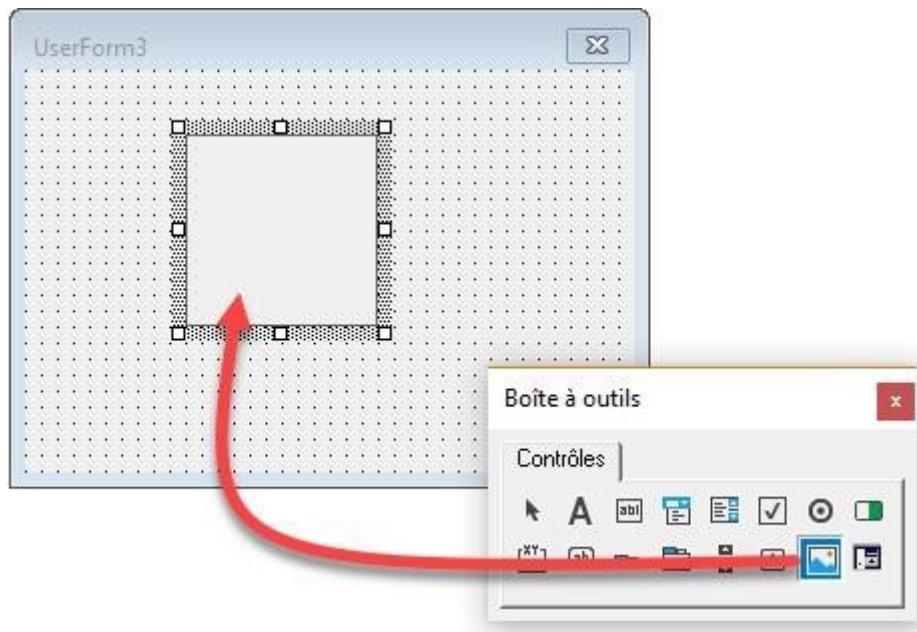
```
Private Sub CommandButton1_Click()
    TextBox2.Text = Evaluate(TextBox1.Text)
End Sub
```

Le résultat de la fonction **Evaluate()** est simplement affecté à la propriété **Text** du deuxième **TextBox** pour afficher le résultat.

Des images dans une boîte de dialogue personnalisée

Le contrôle **Image** permet d'afficher des images dans une boîte de dialogue personnalisée.

Pour insérer un contrôle **Image**, sélectionnez l'outil **Image** dans la boîte à outils, puis cliquez dans la boîte de dialogue personnalisée :



Vous utiliserez les propriétés suivantes :

- **AutoSize** pour adapter la taille de contrôle **Image** à l'image.
- **Picture** pour affecter une image au contrôle **Image**.

Pour affecter une image au contrôle **Image**, vous affecterez la fonction **LoadPicture()** à la propriété **Picture** du contrôle:

```
Image1.Picture = LoadPicture("chemin et nom de l'image")
```

A titre d'exemple, nous allons afficher une image dans une boîte de dialogue personnalisée.

Lorsque l'utilisateur cliquera dessus, une deuxième image sera affichée. Lorsqu'il cliquera à nouveau dessus, la première image sera affichée. Ainsi de suite... Nous utiliserons les deux images suivantes :



Créez une nouvelle boîte de dialogue personnalisée.

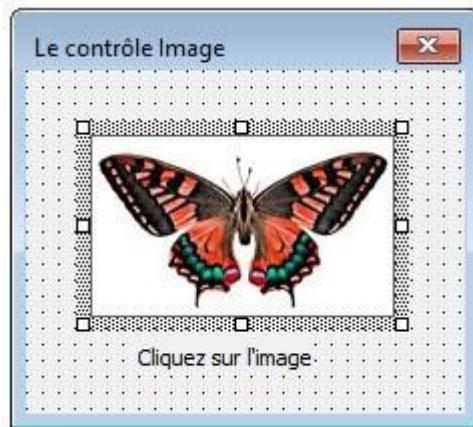
A l'aide de la fenêtre **Propriétés**, affectez le texte "Le contrôle Image" à la propriété **Caption** de **UserForm1**.

Insérez un contrôle **Label**. Cliquez sur ce contrôle. A l'aide de la fenêtre **Propriétés**, affectez le texte "Cliquez sur l'image" à sa propriété **Caption**.

Insérez un contrôle **Image**. Cliquez sur ce contrôle. A l'aide de la fenêtre **Propriétés**, initialisez à **True** sa propriété **AutoSize**. Cliquez sur **(Image bitmap)** en face de la propriété **Picture** et affectez-lui l'image du premier papillon :



Voici ce que vous devriez obtenir :

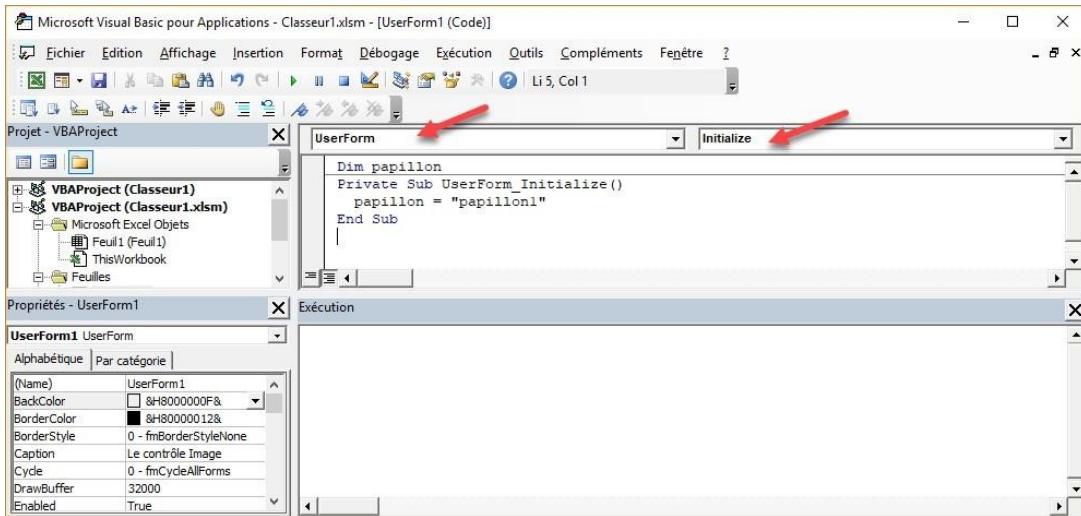


Basculez sur la fenêtre **Code** en appuyant sur la touche de fonction *F7*.

Pour savoir quelle image est affichée dans le contrôle Image, nous allons utiliser une variable que nous appellerons **papillon** :

```
Dim papillon
```

Selectionnez **UserForm** dans la première liste déroulante et **Initialize** dans la deuxième et définissez le code suivant pour mémoriser que le premier papillon est affiché au démarrage de l'application :



Voici le code de la procédure **UserForm_Initialize()** :

```
Private Sub UserForm_Initialize()
    papillon =
    "papillon1"
End Sub
```

Sélectionnez **Image1** dans la première liste déroulante et **MouseUp** dans la seconde, puis complétez la procédure **Image1_MouseUp()** comme ceci :

```
Private Sub Image1_MouseUp(ByVal Button As Integer, ByVal Shift As Integer,
    ByVal X As Single, ByVal Y As Single)

    If papillon = "papillon1" Then

        papillon = "papillon2"

        Image1.Picture = LoadPicture("C:\data\mediaforma\Images\papillon2.jpg")

    Else
        papillon = "papillon1"

        Image1.Picture = LoadPicture("C:\data\mediaforma\Images\papillon1.jpg")
    End If

End Sub
```

Lorsque l'utilisateur clique sur l'image, un événement **MouseUp** est généré. Si la variable **papillon** vaut **papillon1**, cela signifie que le premier papillon est affiché. Dans ce cas, la variable papillon est initialisée à **papillon2** et le deuxième papillon est affiché à la place du premier :

```
If papillon = "papillon1" Then

    papillon = "papillon2"
```

```
Image1.Picture = LoadPicture("C:\data\mediaforma\Images\papillon2.jpg")
```

Si la variable **papillon** est différente de **papillon1**, cela signifie que le deuxième papillon est affiché. Dans ce cas, la variable **papillon** est initialisée à **papillon1** et le premier papillon est affiché à la place du deuxième :

```
Else  
    papillon = "papillon1"  
  
    Image1.Picture = LoadPicture("C:\data\mediaforma\Images\papillon1.jpg")  
End If
```

Tracé d'un graphique en VBA

Cette section va vous montrer comment créer un graphique à partir de données numériques.

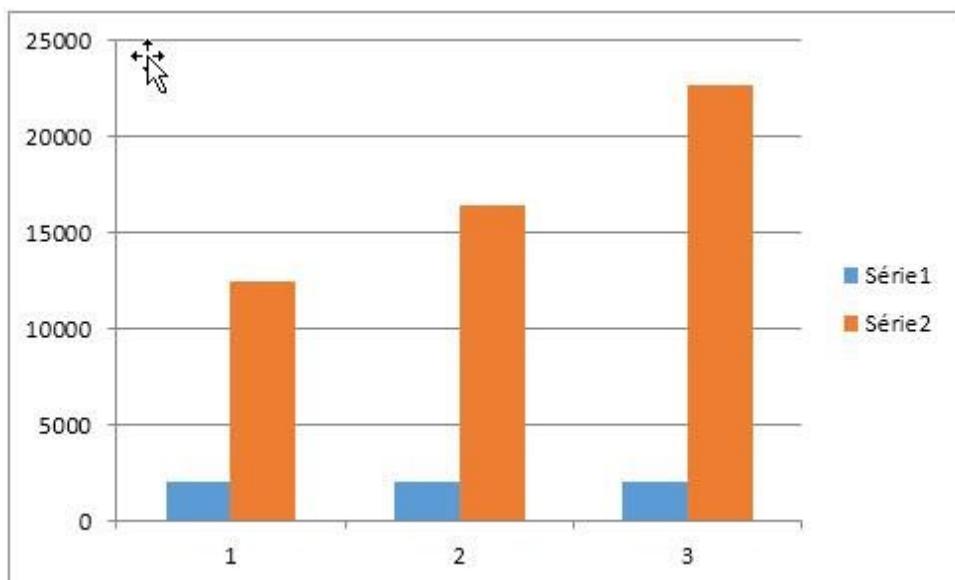
Nous allons partir de ces données :

	A	B	C	D
1	2015	2016	2017	
2	12435	16435	22647	
3				
4				

Ces instructions :

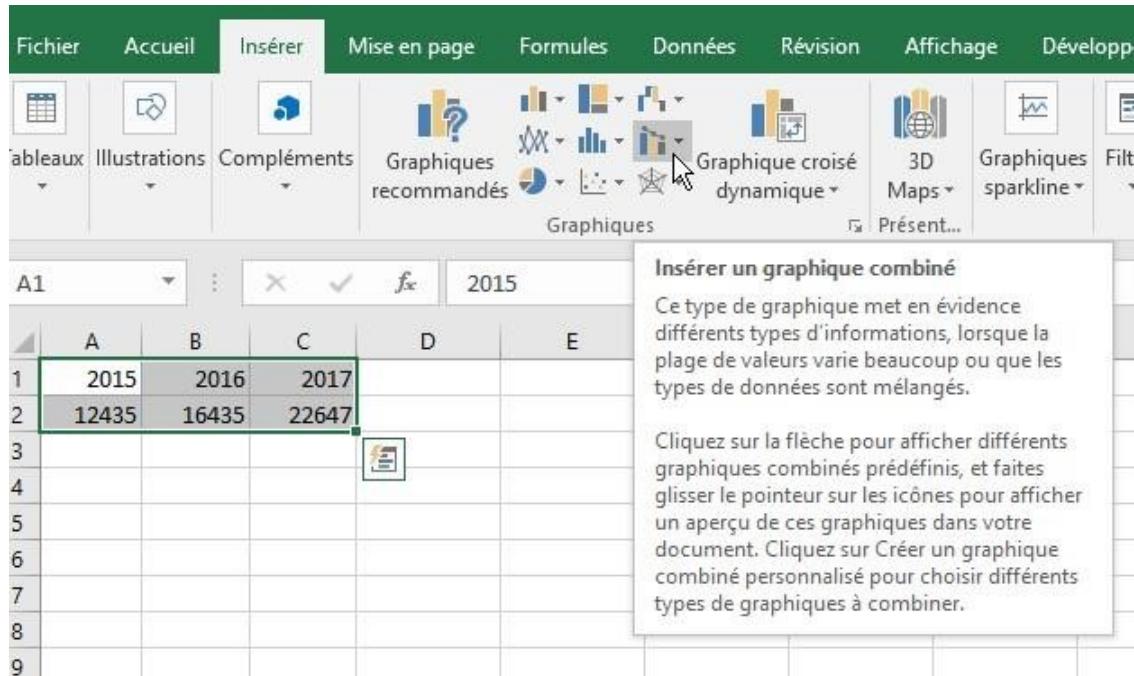
```
Range("A1:C2").Select  
ActiveSheet.Shapes.AddChart.Select
```

Produisent le résultat suivant :



Vous voulez un autre type de graphique ? Utilisez l'enregistreur de macros.

Selectionnez la plage A1-C3, basculez sur l'onglet Développeur du ruban et cliquez sur l'icône Enregistrer une macro dans le groupe Code. Basculez sur l'onglet Insérer du ruban et insérez le graphique qui vous convient. Ici par exemple, nous choisissons un graphique combiné :

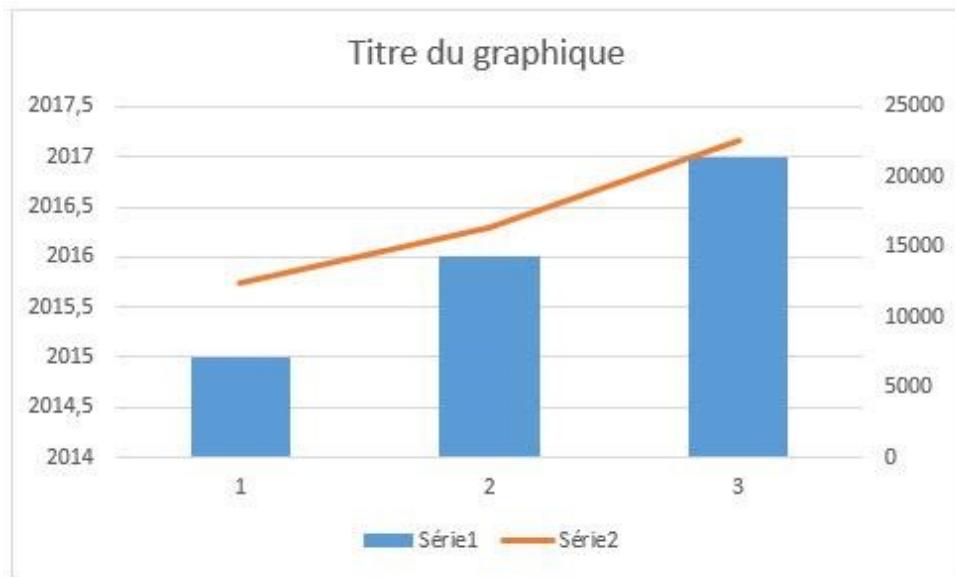


Arrêtez l'enregistrement de la macro en cliquant sur l'icône Arrêter l'enregistrement dans le groupe Code, sous l'onglet Développeur du ruban. Voici le code qui a été généré :

```
ActiveSheet.Shapes.AddChart2(322, xlColumnClustered).Select  
ActiveChart.FullSeriesCollection(1).ChartType = xlColumnClustered  
ActiveChart.FullSeriesCollection(2).ChartType = xlLine  
ActiveChart.FullSeriesCollection(2).AxisGroup = 2
```

Complétez ce code en ajoutant au début l'instruction qui sélectionne la plage A1-C2 :

```
Range("A1:C2").Select  
ActiveSheet.Shapes.AddChart2(322, xlColumnClustered).Select  
ActiveChart.FullSeriesCollection(1).ChartType = xlColumnClustered  
ActiveChart.FullSeriesCollection(2).ChartType = xlLine  
ActiveChart.FullSeriesCollection(2).AxisGroup = 2 Et  
le tour est joué :
```



N'hésitez pas à utiliser l'enregistreur de macros en renfort du VBA. Le code généré sera directement utilisable.

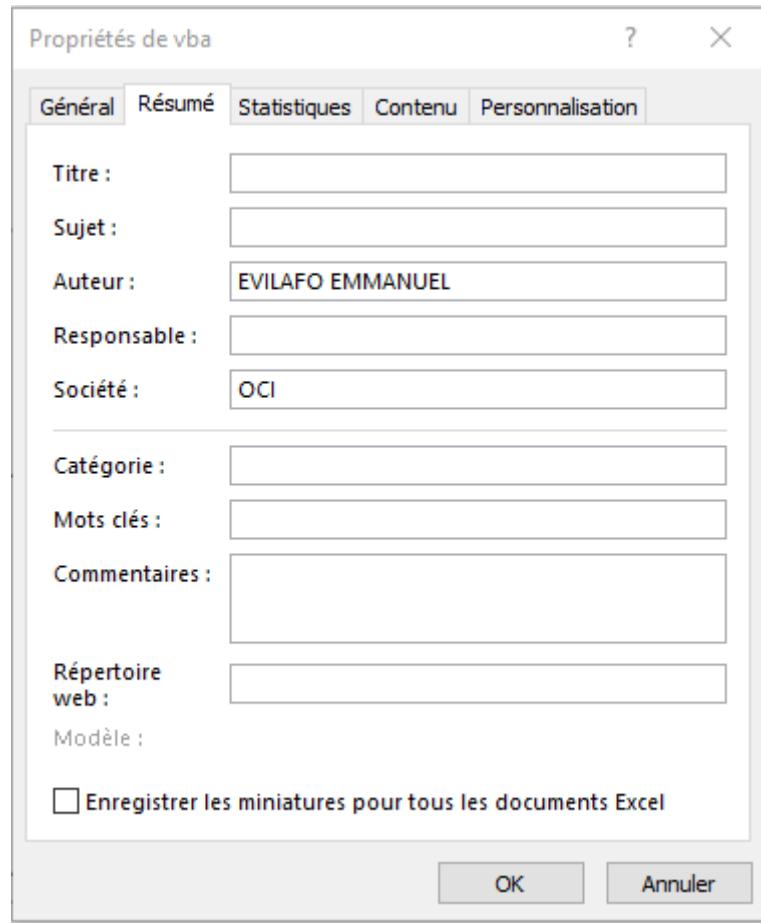
Agir sur les propriétés d'un classeur

Tous les classeurs possèdent des propriétés qui donnent des informations complémentaires sur son contenu, son auteur ou sa catégorie.

Pour accéder aux propriétés d'un classeur Excel 2016, basculez sur l'onglet **Fichier** dans le ruban puis cliquez sur **Informations**. Dans la vue backstage, cliquez sur **Propriétés** puis sur **Propriétés avancées** :



La boîte de dialogue des propriétés s'affiche et donne accès aux principales propriétés du classeur sous l'onglet **Résumé** :

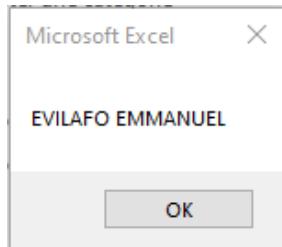


Toutes ces propriétés peuvent être lues et modifiées en VBA. Pour cela, vous préciserez la propriété à laquelle vous voulez accéder dans la fonction **ActiveWorkbook.BuiltinDocumentProperties("p")** (où p est la propriété concernée). Voici quelques-unes des propriétés utilisables :

Nom de la propriété	Equivalent dans la boîte de dialogue des propriétés
Title	Titre
Subject	Sujet
Author	Auteur
Manager	Responsable
Company	Société
Keywords	Mots clés
Comments	Commentaires

Ainsi par exemple, cette instruction affichera le nom de l'auteur du classeur dans une boîte de dialogue :

```
MsgBox ActiveWorkbook.BuiltinDocumentProperties("Author")
```



Où encore, cette instruction affectera la chaîne “Cette propriété a été définie en VBA” à la propriété **Subject** du classeur :

```
ActiveWorkbook.BuiltinDocumentProperties("Subject") = "Cette propriété a  
été définie en VBA"
```

Pour accéder à toutes les propriétés du classeur, il n'est pas nécessaire de connaître leur nom : vous pouvez parcourir la collection **BuiltinDocumentsProperties** avec une boucle **For Each**. Ici par exemple, nous affichons le nom des propriétés dans la colonne 1 et les valeurs correspondantes dans la colonne 2 :

```
On Error Resume Next  
i = 1  
  
Worksheets(1).Activate  
  
For Each p In ActiveWorkbook.BuiltinDocumentProperties  
  
    Cells(i, 1) = p.Name  
  
    Cells(i, 2) = p.Value  
  
    i = i + 1  
Next
```

Remarque

L'instruction **On Error Resume Next** évite l'arrêt du code et l'affichage d'un message d'erreur lorsqu'une propriété n'est pas définie.

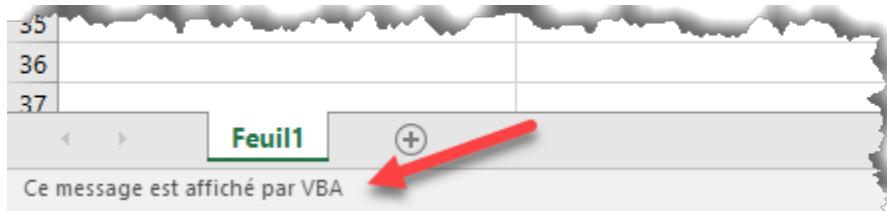
Voici le résultat :

	A	B
1	Title	
2	Subject	Cette propriété a été définie en VBA
3	Author	EVILAFO EMMANUEL / [Consultant]
4	Keywords	
5	Comments	
6	Template	
7	Last author	EVILAFO MMANUEL
8	Revision number	
9	Application name	Microsoft Excel
10	Last print date	
11	Creation date	02/01/2025 16:00
12	Last save time	02/01/2025 16:03
13	Total editing time	
14	Number of pages	
15	Number of words	
16	Number of characters	
17	Security	0
18	Category	
19	Format	
20	Manager	
21	Company	OCI
22	Number of bytes	
23	Number of lines	
24	Number of paragraphs	
25	Number of slides	
26	Number of notes	
27	Number of hidden Slides	
28	Number of multimedia clips	
29	Hyperlink base	
30	Number of characters (with spaces)	
31	Content type	
32	Content status	
33	Language	
34	Document version	

Afficher un message dans la barre d'état

VBA permet d'afficher du texte dans la barre d'état d'Excel *via* la propriété **StatusBar** de l'objet **Application**. A titre d'exemple, cette instruction :

```
Application.StatusBar = "Ce message est affiché par VBA"  
Produit l'affichage suivant dans la barre d'état :
```



Bien entendu, vous pouvez afficher la valeur d'une propriété dans la barre d'état. Par exemple, cette instruction affiche la propriété **Author** de l'objet **Application** du classeur dans la barre d'état, précédée du libellé "Auteur : " :

```
Application.StatusBar = "Auteur : " &  
ActiveWorkbook.BuiltinDocumentProperties("Author")
```

Ou encore, le contenu d'une cellule :

```
Application.StatusBar = "La cellule B2 vaut " & Cells(2,2)
```

Si la barre d'état n'est pas visible chez vous, vous pouvez la faire apparaître en affectant la valeur **True** à la propriété **DisplayStatusBar** de l'objet **Application** :

```
Application.DisplayStatusBar = True
```

Dim et Set

Dans l'article "[Variables et constantes](#)", vous avez appris à déclarer vos variables avec l'instruction **Dim** :

```
Dim entier As Integer
```

```
Dim texte As String
```

Pour affecter une valeur aux variables que vous venez de définir, il suffit d'utiliser le signe "**=**" :

```
entier = 12  
texte = "Un  
texte"
```

L'instruction **Dim** permet également de définir des variables qui contiendront des objets. Par exemple :

```
Dim feuille as Worksheet
```

Cette instruction définit la variable **feuille** de type **Worksheet**. Pour pouvoir travailler avec la variable **feuille**, vous allez lui affecter une feuille avec l'instruction **Set** :

```
Set feuille = Sheets("Feuill1")
```

Ou encore :

```
Set feuille = Sheets(1)
```

Vous utiliserez systématiquement l'instruction **Set** pour affecter un objet à une variable. Voici quelques exemples :

```
Dim plage As Range
```

```
Set plage = Range("A2:A9") 'plage représente la plage A2:A9
```

```
Dim wb As Workbook
```

```
Set wb = ActiveWorkbook 'wb représente le classeur actif
```

```
Dim wb2 As Workbook
```

```
Set wb2 = Workbooks.Add 'Ajout d'un classeur
```

Lorsque vous avez défini un objet, pensez à le libérer à la fin de la procédure en lui affectant la valeur **Nothing** :

```
Set feuille = Nothing
```

```
Set plage = Nothing
```

```
Set wb = Nothing
```

```
Set wb2 = Nothing
```

Définir un nouveau classeur, une nouvelle feuille

Pour créer un nouveau classeur, utilisez les instructions suivantes :

```
Dim classeur As Workbook  
Set classeur = Workbooks.Add
```

Pour insérer une feuille dans le classeur en cours d'utilisation, plusieurs instructions peuvent être utilisées, en fonction de la position de l'insertion.

Ajout d'une feuille avant la feuille active :

```
Sheets.Add.Name = "NouvelleFeuille"
```

Ajout d'une feuille en dernière position :

```
Sheets.Add(After:=Worksheets(Worksheets.Count)).Name = "DerniereFeuille"
```

Ajout d'une feuille en première position :

```
Sheets.Add(Before:=Worksheets(1)).Name = "PremiereFeuille"
```

Ajout d'une feuille en troisième position :

```
Sheets.Add(After:=Worksheets(2)).Name = "TroisiemeFeuille"
```

Ajouter un graphique personnalisé

Voici un exemple de code :

```
Dim feuille As Worksheet  
  
Dim graphique As ChartObject  
  
Set feuille = Sheets("Feuill1")  
  
Set graphique = feuille.ChartObjects.Add(60, 50, 500, 300)  
  
With graphique.Chart  
    .ChartType = xlLineMarkers  
    .SeriesCollection.NewSeries  
  
    With .SeriesCollection(1)  
        .Values = feuille.Range("A2:C2")  
        .XValues = feuille.Range("A1:C1")  
    End With  
  
End With  
  
Set graphique = Nothing  
  
Set feuille = Nothing
```

Les deux premières instructions définissent les variables feuille et graphique qui représenteront respectivement la feuille dans laquelle le graphique sera inséré et le graphique à insérer :

```
Dim feuille As Worksheet
```

```
Dim graphique As ChartObject
```

L'instruction suivante affecte la feuille **Feuil1** à la variable **feuille** :

```
Set feuille = Sheets("Feuil1")
```

L'instruction suivante ajoute à la feuille **Feuil1** un graphique :

- de largeur 500 pixels ;
- de hauteur 500 pixels ;
- dont l'angle supérieur gauche se trouve à 60 pixels du bord gauche et à 50 pixels du bord supérieur.

Pour faciliter l'écriture du code, ce graphique est affecté à la variable **graphique** :

```
Set graphique = feuille.ChartObjects.Add(60, 50, 500, 300)
```

L'instruction **With ... End With** suivante définit les caractéristiques du graphique. On commence par le type du graphique, *via* la propriété **ChartType** :

```
With graphique.Chart
```

```
    .ChartType = xlLineMarkers
```

Comme le montre le tableau suivant, de très nombreux graphiques peuvent être définis dans Excel 2016 :

ChartType	Signification
xl3DArea	Aires 3D
xl3DAreaStacked	Aires 3D empilées
xl3DAreaStacked100	Aires empilées 100 %
xl3DBarClustered	Barres groupées 3D
xl3DBarStacked	Barres empilées 3D
xl3DBarStacked100	Barres empilées 100 % 3D
xl3DColumn	Histogramme 3D
xl3DColumnClustered	Histogramme groupé 3D
xl3DColumnStacked	Histogramme empilé 3D
xl3DColumnStacked100	Histogramme empilé 100 % 3D
xl3DLine	Courbe 3D
xl3DPie	Secteurs en 3D
xl3DPieExploded	Secteurs éclatés en 3D
xlArea	Aires
xlAreaStacked	Aires empilées
xlAreaStacked100	Aires empilées 100 %

xlBarClustered	Barres groupées
xlBarOfPie	Barres de secteurs
xlBarStacked	Barres empilées
xlBarStacked100	Barres empilées 100 %
xlBubble	Bulles
xlBubble3DEffect	Bulles avec effet 3D
xlColumnClustered	Histogramme groupé
xlColumnStacked	Histogramme empilé
xlColumnStacked100	Histogramme empilé 100 %
xlConeBarClustered	Barres groupées à forme conique
xlConeBarStacked	Barres empilées à forme conique
xlConeBarStacked100	Barres empilées 100 % à forme conique
xlConeCol	Histogramme 3D à forme conique
xlConeColClustered	Histogramme groupé à formes coniques
xlConeColStacked	Histogramme empilé à formes coniques
xlConeColStacked100	Histogramme empilé 100 % à formes coniques
xCylinderBarClustered	Barres groupées à formes cylindriques
xCylinderBarStacked	Barres empilées à formes cylindriques
xCylinderBarStacked100	Barres empilées 100 % à formes cylindriques
xCylinderCol	Histogramme 3D à formes cylindriques
xCylinderColClustered	Histogramme groupé à formes coniques
xCylinderColStacked	Histogramme empilé à formes coniques
xCylinderColStacked100	Histogramme empilé 100 % à formes cylindriques
xDoughnut	Anneau
xDoughnutExploded	Anneau éclaté
xFLine	Courbe
xFLineMarkers	Courbes avec marques
xFLineMarkersStacked	Courbe empilée avec marques

xILineMarkersStacked100	Courbe empilée 100 % avec marques
xILineStacked	Courbe empilée
xILineStacked100	Courbe empilée 100 %
xIPie	Secteurs
xIPieExploded	Secteurs éclatés
xIPieOfPie	Secteurs de secteurs
xIPyramidBarClustered	Histogramme groupé à formes pyramidales
xIPyramidBarStacked	Histogramme empilé à formes pyramidales
xIPyramidBarStacked100	Histogramme empilé 100 % à formes pyramidales
xIPyramidCol	Histogramme 3D à formes pyramidales
xIPyramidColClustered	Histogramme groupé à formes pyramidales
xIPyramidColStacked	Histogramme empilé à formes pyramidales
xIPyramidColStacked100	Histogramme empilé 100 % à formes pyramidales
xIRadar	Radar
xIRadarFilled	Radar plein
xIRadarMarkers	Radar avec marqueurs
xIStockHLC	Max-Min-Clôture
xIStockOHLC	Ouverture-Max-Min-Clôture
xIStockVHLC	Volume-Max-Min-Clôture
xIStockVOHLC	Volume-Ouverture-Max- Min-Clôture
xISurface	Surface 3D
xISurfaceTopView	Surface 3D avec structure apparente
xISurfaceTopViewWireframe	Contour
xISurfaceWireframe	Contour filaire
xIXYScatter	Nuage de points
xIXYScatterLines	Nuages de points avec courbes
xIXYScatterLinesNoMarkers	Nuages de points avec courbes et sans marqueurs
xIXYScatterSmooth	Nuages de points avec courbes lissées
xIXYScatterSmoothNoMarkers	Nuages de points avec courbes lissées et sans marqueurs

L'instruction suivante ajoute une collection de séries :

```
.SeriesCollection.NewSeries
```

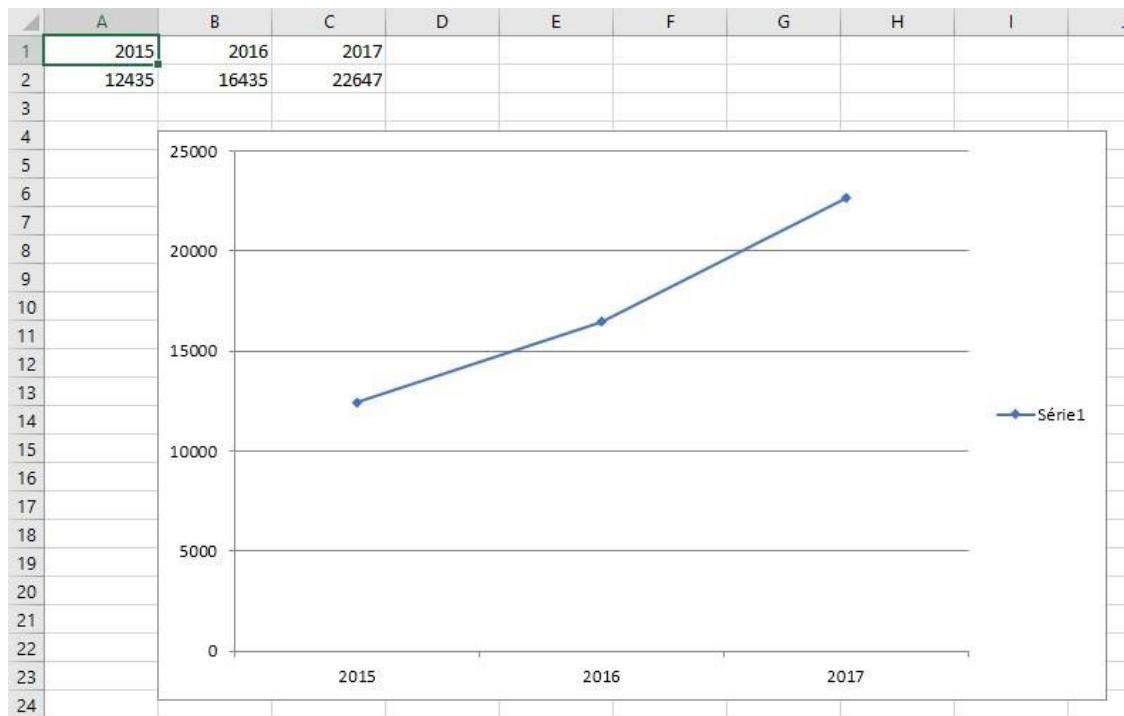
Cette collection est utilisée pour définir les abscisses (**XValues**) et les ordonnées (**Values**) correspondantes :

```
With .SeriesCollection(1)  
    .Values = feuille.Range("A2:C2")  
    .XValues = feuille.Range("A1:C1")  
End With
```

Le code se termine par la suppression des variables objet **graphique** et **feuille** :

```
Set graphique = Nothing  
  
Set feuille = Nothing
```

Voici un exemple d'exécution :



Changer la couleur de certaines cellules en fonction de leur valeur

Cette section va vous montrer comment changer la couleur de certaines cellules en fonction de leur valeur. Nous allons partir de la feuille suivante :

	A	B	C	D	E	F	G
1	Départements et régions	Population moyenne	Mariages	Divorces	Naissances	Décès	Accroissement naturel
2	Abidjan	22211471	10002	5117	29134	13487	7
3	Bouaké	1384758	5064	2662	19708	8490	8,1
4	Yamoussoukro	1422989	4429	2466	19525	8433	7,8
5	Divo	1273257	4403	2219	18903	7384	9
6	Daloa	1600296	5734	2717	24675	9547	9,4
7	San-Pedro	1568318	5824	2479	29471	8272	13,5
8	Assinie	1367273	4891	2374	21566	7953	10
9	Korhogo	1206326	4480	2105	19737	6940	10,6
10							

A titre d'exemple, nous allons parcourir les cellules de la colonne G. Lorsque la valeur d'une de ces cellules sera supérieure ou égale à **10,5**, la ligne correspondante sera colorée en orange. Dans le cas contraire, la ligne correspondante sera colorée en vert.

Voici le code utilisé :

```
Dim i As Integer  
  
For i = 2 To 9  
  
    If Cells(i, 7) >= 10.5 Then  
  
        Range(Cells(i, 1), Cells(i, 7)).Interior.Color = RGB(255, 128, 128)  
    Else  
  
        Range(Cells(i, 1), Cells(i, 7)).Interior.Color = RGB(128, 255, 128)  
    End If  
  
Next i
```

Une boucle parcourt les cellules de la plage **G2:G9**. Si la valeur contenue dans une de ces cellules est supérieure ou égale à **10,5** :

```
For i = 2 To 9  
  
    If Cells(i, 7) >= 10.5 Then
```

La plage comprise entre les colonnes A et G de la ligne concernée est colorée en orange :

```
Range(Cells(i, 1), Cells(i, 7)).Interior.Color = RGB(255, 128, 128)
```

Dans le cas contraire, cette même plage est colorée en vert :

```
Else
```

```
    Range(Cells(i, 1), Cells(i, 7)).Interior.Color = RGB(128, 255, 128)
```

Voici le résultat :

	A	B	C	D	E	F	G
1	Départements et régions	Population moyenne	Mariages	Divorces	Naissances	Décès	Accroissement naturel
2	Abidjan	22211471	10002	5117	29134	13487	7
3	Bouaké	1384758	5064	2662	19708	8490	8,1
4	Yamoussoukro	1422989	4429	2466	19525	8433	7,8
5	Divo	1273257	4403	2219	18903	7384	9
6	Daloa	1600296	5734	2717	24675	9547	9,4
7	San-Pedro	1568318	5824	2479	29471	8272	13,5
8	Assinie	1367273	4891	2374	21566	7953	10
9	Korhogo	1206326	4480	2105	19737	6940	10,6
10							

Exécuter une procédure à une certaine heure

Vous pouvez demander à VBA d'exécuter une procédure à une heure donnée en utilisant la méthode **Application.OnTime** :

```
Application.OnTime heure, "proc"
```

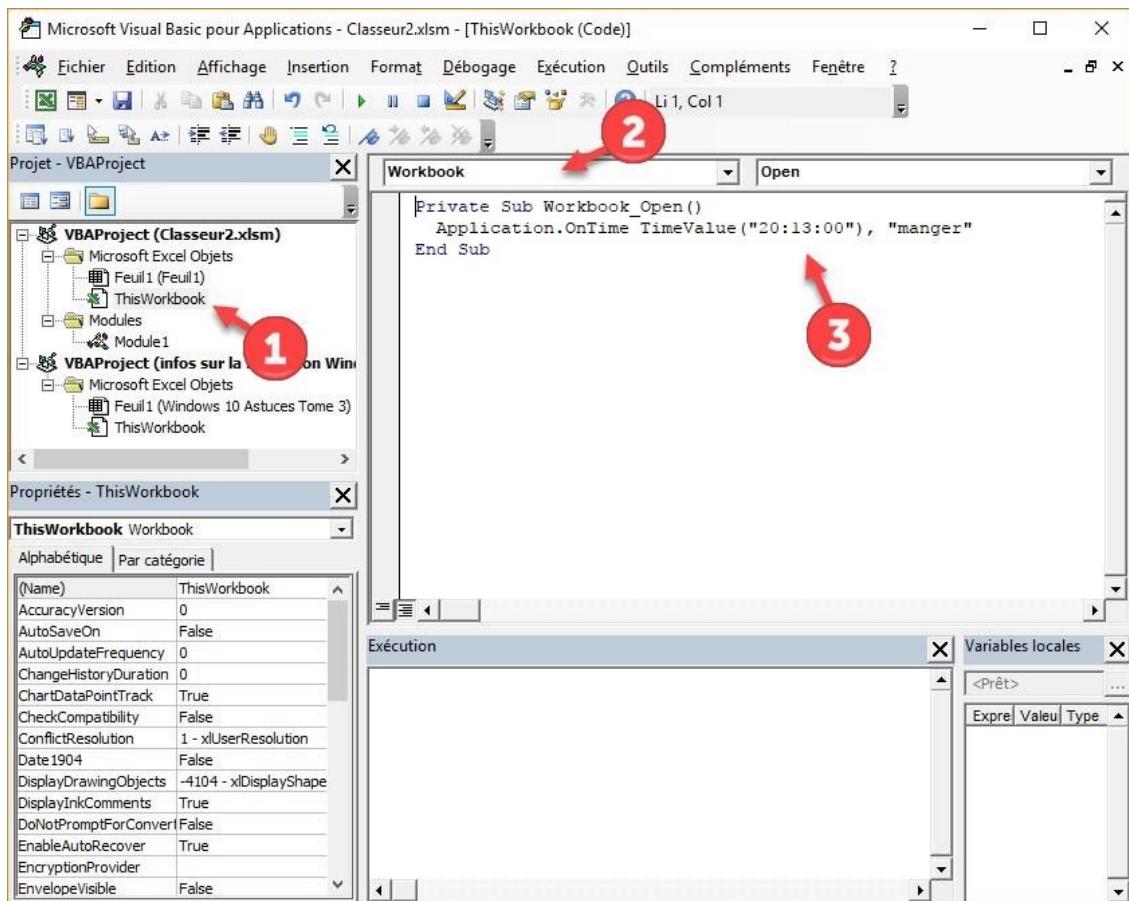
Où **heure** est un objet **Time** qui définit l'heure d'exécution de la procédure **proc**.

Exécuter une procédure à une heure fixe

Vous appellerez la méthode **Application.OnTime** dans la procédure **Workbook_Open()**.

Ouvrez la fenêtre Microsoft Visual Basic pour Applications du classeur concerné.

Doublecliquez sur **ThisWorkbook** dans la fenêtre **Projet** (1) et sélectionnez **Workbook** dans la liste déroulante **Objet** (2). La procédure **Workbook_Open()** est automatiquement créée. Il ne vous reste plus qu'à la compléter (3) :



Lorsque vous ouvrirez le classeur, le code placé dans la procédure **Workbook_Open()** sera automatiquement exécuté. Voici le code utilisé :

```
Private Sub Workbook_Open()

    Application.OnTime TimeValue("12:30:00"), "manger"

End Sub
```

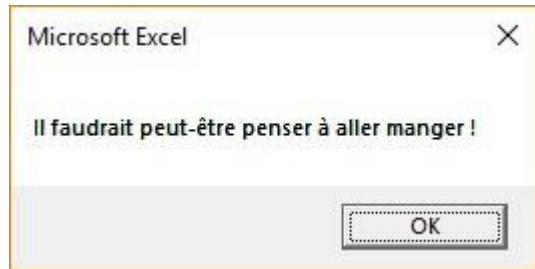
Ici, nous utilisons la fonction **TimeValue()** pour obtenir un objet **Time** à partir d'une chaîne au format **hh:mm:ss**. Cet objet définit l'heure d'exécution de la procédure **manger()**.

Il ne reste plus qu'à définir la procédure **manger()**. Sous **Module**, double-cliquez sur **Module1** et définissez la procédure **manger()** :

```
Sub manger()

    MsgBox "Il faudrait peut-être penser à aller manger !"
End Sub
```

A 12 heures 30, une boîte de message rappellera que c'est l'heure du repas :



Exécuter une procédure un certain laps de temps après l'ouverture du classeur

En utilisant la procédure **Application.OnTime**, vous pouvez également exécuter une procédure un certain laps de temps après l'ouverture du classeur.

La procédure **Workbook.Open()** est la très proche de celle utilisée dans l'exemple précédent, si ce n'est qu'ici, on ajoute le délai (**TimeValue**) à l'heure actuelle (**Now**). Dans cet exemple, la procédure **alerte()** s'exécutera une minute après l'ouverture du classeur :

```
Private Sub Workbook_Open()  
  
    Application.OnTime Now + TimeValue("00:01:00"), "alerte"  
  
End Sub
```

Voici le code de la fonction **alerte()** :

```
Sub alerte()  
  
    MsgBox "Le classeur est ouvert depuis une minute."  
  
End Sub
```

Exécution d'une procédure à l'appui sur une touche ou une combinaison de touches

VBA est en mesure d'exécuter une procédure lorsque l'utilisateur appuie sur une touche ou une combinaison de touches. Pour cela, vous utiliserez la procédure **Application.OnKey** :

```
Application.OnKey "touche", "proc"
```

Où **touche** est la touche ou la combinaison de touches qui déclenche l'exécution de la procédure **proc**.

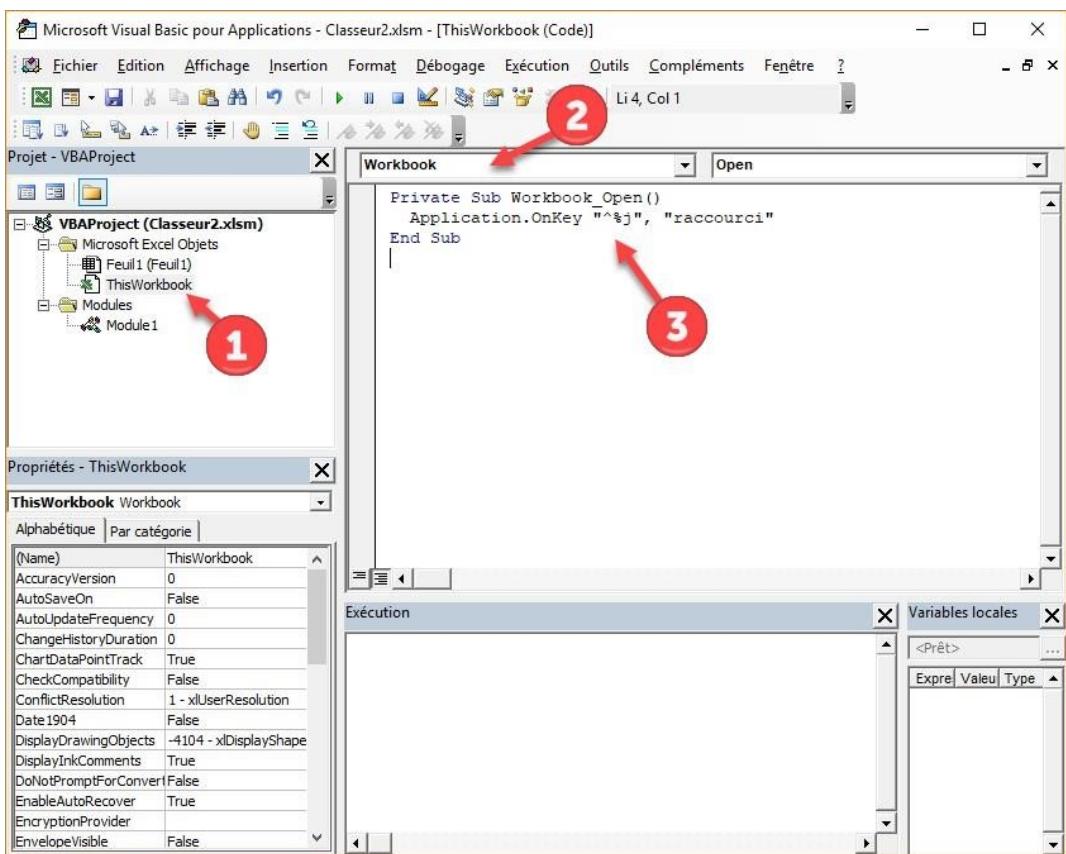
Le tableau ci-après donne un aperçu de la syntaxe à utiliser.

Touche	Code de la touche
<i>Majuscule</i>	+
<i>Contrôle</i>	^
<i>Alt</i>	%
<i>Suppr</i>	{DELETE}
<i>Retour Arrière</i>	{BACKSPACE}
<i>Verr Num</i>	{NUMLOCK}
<i>Verr Maj</i>	{CAPSLOCK}
<i>Arrêt Defil</i>	{SCROLLLOCK}
<i>Page Suivante</i>	{PGDN}
<i>Page précédente</i>	{PGUP}
<i>Haut</i>	{UP}
<i>Bas</i>	{DOWN}
<i>Gauche</i>	{LEFT}
<i>Droite</i>	{RIGHT}
<i>Origine</i>	{HOME}
<i>Fin</i>	{END}
<i>F1 à F12</i>	{F1} à {F12}
<i>Entrée</i>	{ENTER}
<i>Echap</i>	{ESC}
<i>Insertion</i>	{INSERT}
<i>Impr écran</i>	{PRTSC}
<i>Tabulation</i>	{TAB}

Vous appellerez la méthode **Application.OnKey** dans la procédure **Workbook_Open()**.

Ouvrez la fenêtre Microsoft Visual Basic pour Applications du classeur concerné.

Doublecliquez sur **ThisWorkbook** dans la fenêtre **Projet** (1) et sélectionnez **Workbook** dans la liste déroulante **Objet** (2). La procédure **Workbook_Open()** est automatiquement créée. Il ne vous reste plus qu'à la compléter (3). Ici par exemple, la procédure **raccourci()** est exécutée lorsque l'utilisateur appuie sur *Contrôle + Alt + j* :



Voici le code utilisé :

```
Private Sub Workbook_Open()

    Application.OnKey "^%j", "raccourci"

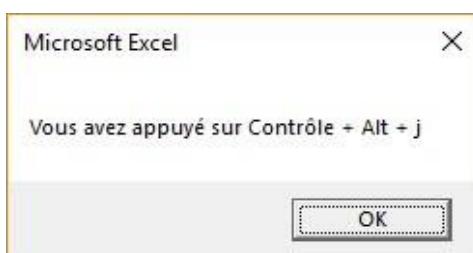
End Sub
```

Il ne reste plus qu'à définir la procédure **raccourci()**. Sous **Module**, double-cliquez sur **Module1** et définissez la procédure **raccourci()** :

```
Sub raccourci()

    MsgBox "Vous avez appuyé sur Contrôle + Alt + J"
End Sub
```

Chaque fois que l'utilisateur appuie simultanément sur les touches *Contrôle*, *Alt* et *j*, une boîte de dialogue s'affiche :



Copier, renommer et supprimer un fichier

VBA est en mesure d'effectuer des opérations élémentaires sur des fichiers. Cette section va vous montrer comment copier, renommer et supprimer un fichier.

Copier un fichier

Pour copier un fichier, vous utiliserez l'instruction **FileCopy** :

```
FileCopy "source", "destination"
```

Où **source** est le chemin complet du fichier à copier et **destination** est le chemin complet de la copie.

Par exemple, pour copier le fichier **Paye.xlsm** qui se trouve dans le dossier **c:\data\encours** dans le fichier **Paye-Janvier.xlsm** du même dossier, vous utiliserez l'instruction suivante :

```
FileCopy "c:\data\encours\Paye.xlsm", "c:\data\encours\Paye-Janvier.xlsm"
```

Renommer un fichier

Pour renommer un fichier, vous utiliserez l'instruction **Name As** :

```
Name "ancien" as "nouveau"
```

Où **ancien** est le chemin complet du fichier à renommer et **nouveau** est le chemin complet du fichier renommé.

Par exemple, pour renommer le fichier **Paye-Janvier.xlsm** qui se trouve dans le dossier **c:\data\encours** en **Paye-Fevrier.xlsm**, vous utiliserez l'instruction suivante :

```
Name "c:\data\encours\Paye-Janvier.xlsm" As "c:\data\encours\Paye-Fevrier.xlsm"
```

Supprimer un fichier

Pour supprimer un fichier, vous utiliserez l'instruction **Kill** :

```
Kill "fichier"
```

Où **fichier** est le chemin complet du fichier à supprimer.

Par exemple, pour supprimer le fichier **Paye-Fevrier.xlsm** qui se trouve dans le dossier **c:\data\encours**, vous utiliserez l'instruction suivante :

```
Kill "c:\data\encours\Paye-Fevrier.xlsm"
```

Lister les fichiers contenus dans un dossier

Cette section va vous montrer comment lister les fichiers contenus dans un dossier quelconque dans une feuille de calcul. Ici, les fichiers seront listés dans les cellules de la colonne **A** de la feuille de calcul **Feuil1**.

Pour cela, nous utiliserons la fonction **Dir()** pour parcourir le dossier :

```
Dim Fichier As String
```

```
Fichier = Dir("chemin")
```

Où **chemin** représente le chemin du dossier à examiner. Par exemple **c:\dossier\sousdossier**.

Si nécessaire, vous pouvez préciser le modèle des fichiers recherchés à la suite du chemin. Par exemple, **c:\dossier\sous-dossier*.docx** recherchera les fichiers d'extension **docx** dans le dossier **c:\dossier\sous-dossier**.

Voici le code utilisé :

```
Dim Dossier As String, Fichier As String, i As Integer
```

```
Dossier = "C:\data\encours\"  
i =  
0
```

```
Fichier = Dir(Dossier)
```

```
Do While Fichier <> ""
```

```
    i = i + 1
```

```
    Sheets("Feuil1").Range("A" & i) = Fichier
```

```
    Fichier = Dir
```

```
Loop
```

La première ligne définit les variables utilisées dans le programme.

```
Dim Dossier As String, Fichier As String, i As Integer
```

La deuxième ligne affecte le dossier dont on désire lister les fichiers à la variable **Dossier** :

```
Dossier = "C:\data\encours\"
```

La troisième ligne initialise la variable compteur qui sera utilisée pour copier le nom des fichiers dans la feuille de calcul :

```
i = 0
```

La quatrième ligne utilise la fonction **Dir()** pour rechercher les fichiers dans le dossier spécifié en argument :

```
Fichier = Dir(Dossier)
```

Une boucle **Do While** parcourt les fichiers listés dans la variable **Fichier** jusqu'au dernier :

```
Do While Fichier <> ""
```

La variable compteur est incrémentée, puis le nom du fichier courant est copié dans la cellule de colonne **A** et de ligne **i** :

```
i = i + 1  
  
Sheets("Feuill1").Range("A" & i) = Fichier
```

Pour passer au fichier suivant, il suffit d'affecter la fonction **Dir** à la variable **Fichier** :

```
Fichier = Dir
```

```
Loop
```

Voici un exemple d'exécution :

	A	B
1	100.swf	
2	344.php	
3	4911_001.pdf	
4	A faire.txt	
5	accord renego.pdf	
6	Accusé de réception.pdf	
7	addevent.php	
8	affiliation.php	
9	ag prévoir 2025.pdf	
10	agessa régularisation 4e Trimestre 2024.pdf	11
	11	agreg-win81.jpg
11	ajouter-flocons-de-neige-photo-1.jpg	
12	align.htm	
13	appartement G.pdf	
14	asgs.pdf	
15		
16		
17		

Nombre de fichiers contenus dans un dossier

Vous voulez savoir combien de fichiers se trouvent dans un dossier de vos unités de masse ? Vous êtes au bon endroit.

Pour parcourir un dossier, nous affecteront la fonction **Dir()** à une variable **String**, en précisant le chemin du dossier à examiner en paramètre de la fonction. **Dir()** retourne le nom du premier fichier du dossier parcouru. Tant que la valeur renournée n'est pas vide, cela

signifie que tous les fichiers n'ont pas été passés en revue. Dans ce cas, vous incrémenterez une variable compteur et vous passerez au fichier suivant en affectant la fonction **Dir** sans paramètre à la variable **String**. Lorsque la variable **String** sera vide, tous les fichiers auront été parcourus et la variable compteur contiendra le nombre de fichiers du dossier.

Voici le code utilisé :

```
Dim Fichier As String, NbFic As Integer  
  
NbFic = 0  
  
Fichier = Dir("c:\data\encours\")  
  
Do While Fichier <> ""  
  
    NbFic = NbFic + 1  
  
    Fichier = Dir  
  
Loop  
  
MsgBox NbFic
```

Et voici un exemple d'exécution :



Tester si un fichier existe

Cette section va vous montrer comment tester si un fichier quelconque existe.

Vous utiliserez la fonction **Dir()**. Passez-lui le chemin complet du fichier dont vous voulez tester l'existence. Elle retournera : □ Une chaîne vide si le fichier n'est pas trouvé.

Le nom du fichier (sans son dossier) si le fichier est trouvé.

Voici le code utilisé :

```
Dim Fichier As String  
  
Fichier = Dir("c:\data\A lire.txt")
```

```

If Fichier <> "" Then

    MsgBox "Le fichier '" & Fichier & "' existe"

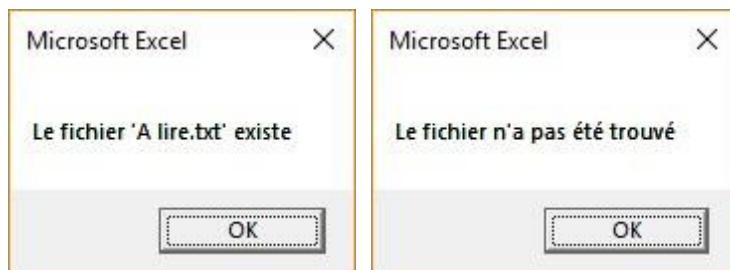
Else

    MsgBox "Le fichier n'a pas été trouvé"

End If

```

Et voici les boîtes de dialogue affichées selon si le fichier existe ou s'il n'existe pas :



Sauvegarde dans un fichier texte

Pour stocker du texte dans un fichier texte, vous ouvrirez ce fichier avec la méthode **Open**, puis vous écrirez une ou plusieurs fois dedans avec la méthode **Print**. Lorsque la totalité du texte aura été écrite, vous fermez le fichier avec la méthode **Close**.

La méthode **Open** peut ouvrir le fichier selon trois modes :

- **Input** : lecture seule.
- **Output** : écriture avec effacement du fichier à chaque ouverture. ☐
- **Append** : écriture avec ajout après la dernière ligne du fichier.

Voici la syntaxe de l'instruction Open :

```
Open Fichier For Input|Output|Append As #f
```

Où :

- **Fichier** est le chemin complet du fichier à ouvrir.
- **Input, Output et Append** sont les trois modes d'ouverture du fichier. ☐ **f** est un numéro de fichier, compris entre **1** et **511**.

Le numéro du fichier ne doit pas être pris par un autre programme. Pour obtenir le premier numéro disponible, vous utiliserez la méthode **FreeFile** :

```
Dim f As Integer  
  
f = FreeFile  
  
Open Fichier For Output As #f
```

Un premier exemple

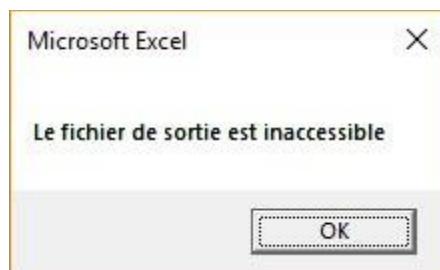
Dans ce premier exemple, une ligne de texte est sauvegardée dans le fichier c:\data\fichiertexte.txt. Une gestion d'erreur est mise en place à l'aide de l'instruction On Error GoTo. Si la création du fichier est impossible (par exemple parce que le dossier destination n'existe pas), un message d'erreur sera affiché :

```
Sub Macro1()  
  
    On Error GoTo Erreur  
  
    Dim Chaine As String  
  
    Dim Fichier As String  
  
    Chaine = "Le texte à sauvegarder"  
  
    Fichier = "c:\data\fichiertexte.txt"  
  
    Dim f As Integer  
  
    f = FreeFile  
  
    Open Fichier For Output As #f  
  
    Print #f, Chaine  
  
    Close #f  
  
    MsgBox "Le texte a été sauvegardé dans: " & Fichier  
    Exit Sub  
  
Erreur:  
  
    MsgBox "Le fichier de sortie est inaccessible"  
  
End Sub
```

Voici le message affiché lorsque l'écriture a eu lieu dans le fichier c:\data\fichiertexte.txt :



Si l'écriture est impossible, un message d'erreur s'affiche :



Copie de la plage A1:A5 dans un fichier texte

Vous allez aller un peu plus loin en copiant dans le fichier texte le contenu des cellules **A1** à **A5** de la feuille de calcul active. La copie se fera à raison d'une cellule par ligne. Nous allons partir de ces données :

	A	B	C
1	Cellule A1		
2	Cellule A2		
3	Cellule A3		
4	Cellule A4		
5	Cellule A5		
6			
7			

Voici le code utilisé :

```
Sub Macro1()

    On Error GoTo Erreur

    Dim Chaine As String

    Dim Fichier As String

    Fichier = "c:\data\sauvegarde.txt"

    Dim f As Integer

    f = FreeFile

    Open Fichier For Output As #f
```

```

For i = 1 To 5

    Print #f, Cells(i, 1)

Next

Close #f

MsgBox "Les cellules ont été sauvegardées dans " & Fichier

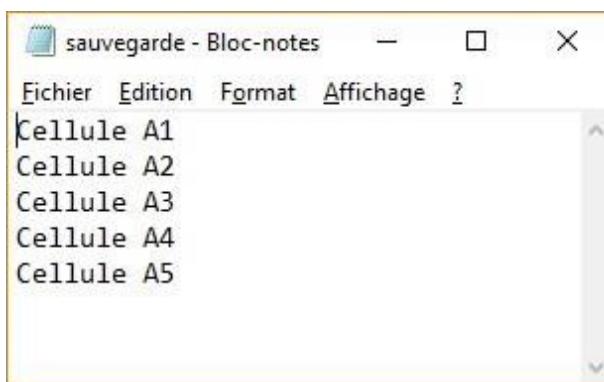
Exit Sub
Erreur:

    MsgBox "Le fichier de sortie est inaccessible"

End Sub

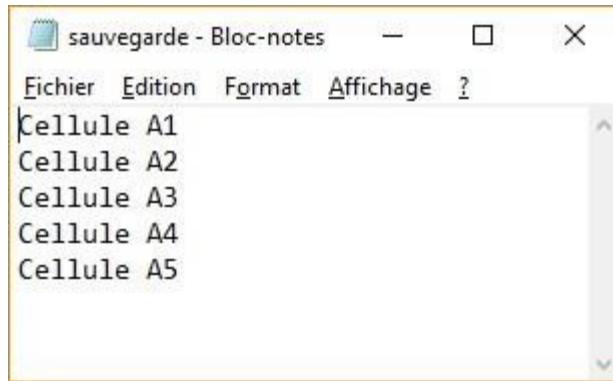
```

Ici, une simple boucle For Next a été utilisée pour parcourir les cellules de la feuille active (Cells(i,1)) et les stocker dans le fichier texte (Print). Voici le fichier texte résultant :



Lecture d'un fichier texte

Dans cette section, vous allez apprendre à accéder à un fichier texte en lecture. Ici, nous allons lire le contenu du fichier texte **sauvegarde.txt**, qui a été défini dans l'article “Sauvegarde dans un fichier texte”. Voici le contenu du fichier sauvegarde.txt :



Les cinq lignes du fichier **sauvegarde.txt** seront copiées dans les cellules **B1** à **B5** pour obtenir ce résultat :

A	B	C
1	Cellule A1	Cellule A1 dans la cellule B1
2	Cellule A2	Cellule A2 dans la cellule B2
3	Cellule A3	Cellule A3 dans la cellule B3
4	Cellule A4	Cellule A4 dans la cellule B4
5	Cellule A5	Cellule A5 dans la cellule B5
6		
7		
8		

Voici le code utilisé :

```
Sub Macro1()

    On Error GoTo Erreur

    Dim Chaine As String

    Dim Fichier As String

    Dim UneLigne As String

    Dim i As Integer

    Fichier = "c:\data\sauvegarde.txt"

    Dim f As Integer
    f =
    FreeFile

    Open Fichier For Input As #f
    i =
    0

    While Not EOF(f)
```

```

i = i + 1

Line Input #f, UneLigne

UneLigne = UneLigne & " dans la cellule B" & i

Cells(i, 2) = UneLigne

Wend

Close #f

Exit Sub
Erreur:

MsgBox "Le fichier de sortie est inaccessible"

End Sub

```

Il n'y a rien de compliqué dans ce code.

Une boucle **While Wend** parcourt le fichier sauvegarde.txt. La boucle prend fin lorsque tout le fichier a été parcouru :

```

While Not EOF(f)
  ...
Wend

```

Les lignes du fichier texte sont lues une par une avec une instruction **Line Input** :

Line Input #f, UneLigne

Le texte lu est complété :

UneLigne = UneLigne & " dans la cellule B" & i

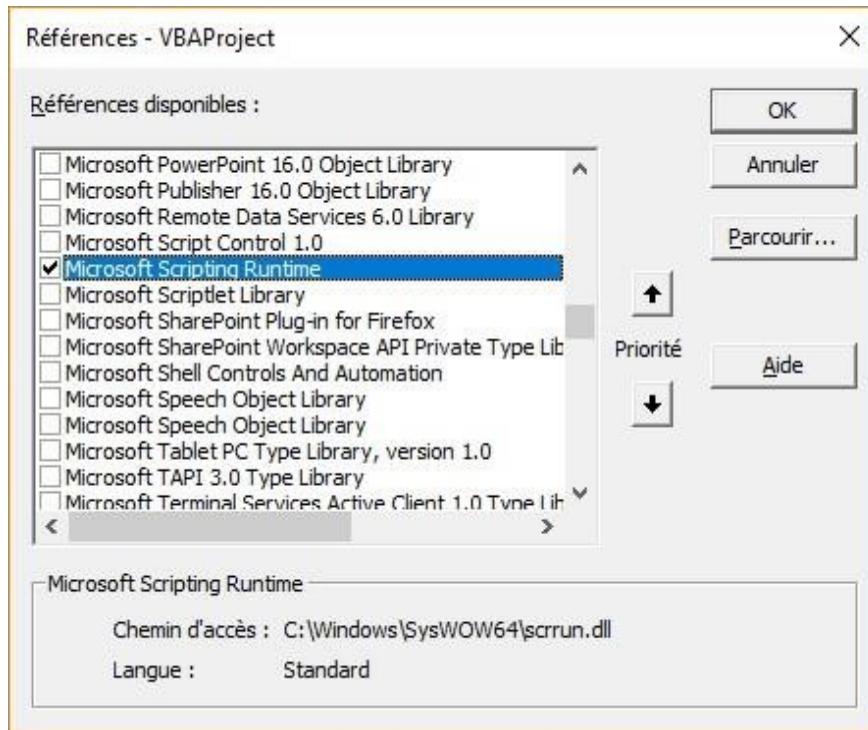
Puis stocké dans la cellule i,2 :

Cells(i, 2) = UneLigne

Manipuler des dossiers

Cette section va vous montrer comment manipuler des dossiers en utilisant des instructions VBA dans Excel. Vous verrez comment créer un dossier, supprimer un dossier, tester si un dossier existe et copier ou déplacer un dossier dans un autre.

Les instructions utilisées s'adresseront à un objet **Scripting.FileSystemObject**. Pour pouvoir créer un tel objet, vous devez définir une référence à la bibliothèque **Microsoft Scripting Runtime**. Lancez la commande **Références** dans le menu **Outils**. La boîte de dialogue **Références** s'affiche. Déplacez-vous dans la zone de liste des références disponibles et cochez la référence **Microsoft Scripting Runtime** :



Validez en cliquant sur **OK**. Vous êtes désormais prêt à manipuler des dossiers via un objet **Scripting.FileSystemObject**.

Créer un dossier

Pour créer un dossier, commencez par créer un objet **Scripting.FileSystemObject** :

```
Dim fs As New Scripting.FileSystemObject
```

Utilisez alors la méthode **CreateFolder** de l'objet **Scripting.FileSystemObject** pour créer un dossier en indiquant son chemin en argument :

```
fs.CreateFolder "chemin du dossier à créer"
```

Attention

Si le chemin comporte plusieurs niveaux, le niveau **N** ne pourra être créé que si le niveau **N-1** existe.

Une fois le dossier créé, libérez la mémoire de l'objet **fs** :

```
Set fs = Nothing
```

Voici un exemple de code complet. Ici on suppose que le dossier **c:\data** existe et on crée le dossier **c:\data\excel** :

```

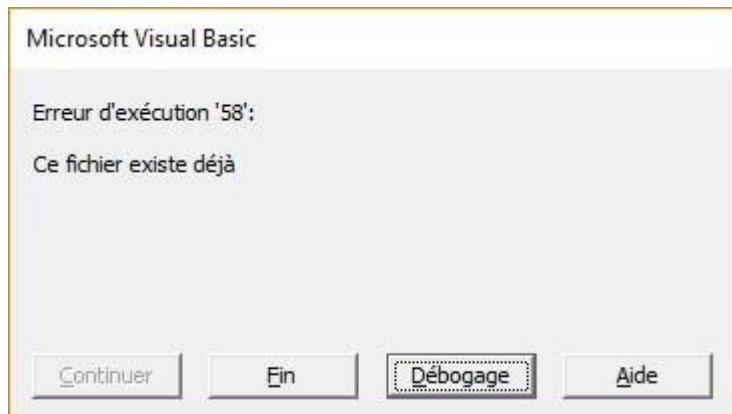
Dim fs As New Scripting.FileSystemObject

fs.CreateFolder "c:\data\excel"

Set fs = Nothing

```

Si le dossier spécifié en argument de la méthode **CreateFolder** existe, une erreur sera levée et une boîte de dialogue peu engageante s'affichera :



Pour améliorer les choses, vous pouvez mettre en place un gestionnaire d'erreurs :

```

On Error GoTo gestionErreurs

Dim fs As New Scripting.FileSystemObject

fs.CreateFolder "c:\data\excel"

Set fs = Nothing

End
gestionErreurs:

    MsgBox "Erreur n° " & Err.Number & vbCrLf & Err.Description
End Sub

```

Si vous essayez de créer un dossier qui existe, la boîte de dialogue suivante s'affichera :



Si vous le souhaitez, vous pouvez créer un dossier sans passer par un objet **Scripting.FileSystemObject** :

```
MkDir "c:\data\excel"
```

Ou encore, en intégrant la gestion d'erreurs :

```
On Error GoTo gestionErreurs  
  
MkDir "c:\data\excel"  
  
End  
gestionErreurs:  
  
    MsgBox "Erreur n° " & Err.Number & vbCrLf & Err.Description
```

Supprimer un dossier

Pour supprimer un dossier, commencez par créer un objet **Scripting.FileSystemObject** :

```
Dim fs As New Scripting.FileSystemObject
```

Indiquez alors le nom du dossier à supprimer à la méthode **DeleteFolder** de l'objet

```
Scripting.FileSystemObject : fs.DeleteFolder "F:\Atelier\Armoire"
```

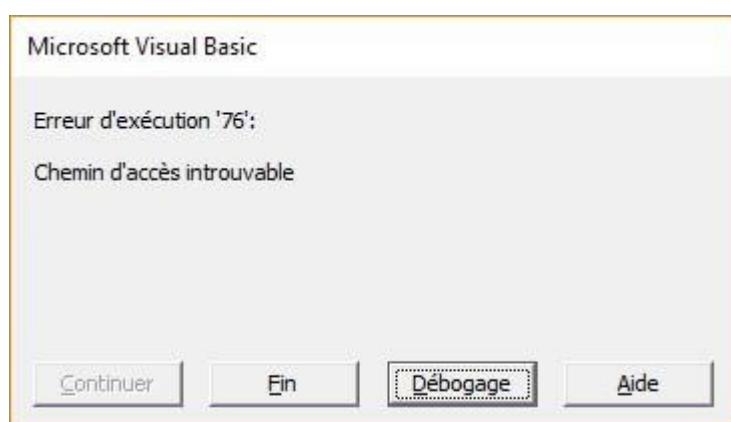
Puis supprimez l'objet **fs** de la mémoire :

```
Set fs = Nothing
```

Voici le code complet, sans gestionnaire d'erreurs :

```
Dim fs As New Scripting.FileSystemObject  
  
fs.DeleteFolder "F:\Atelier\Armoire"  
  
Set GestionFichier = Nothing
```

Si vous essayez de supprimer un dossier inexistant, une erreur VBA est levée :



Pour améliorer les choses, définissez un gestionnaire d'erreurs :

```
On Error GoTo gestionErreurs  
  
Dim fs As New Scripting.FileSystemObject
```

```

fs.DeleteFolder "c:\data\excel"

Set GestionFichier = Nothing

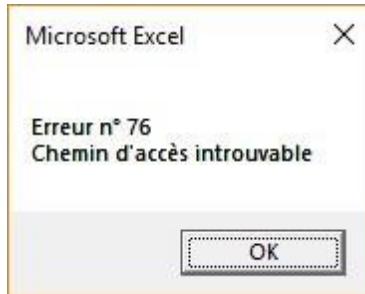
End
gestionErreurs:

MsgBox "Erreur n° " & Err.Number & vbCrLf & Err.Description

End Sub

```

Voici la boîte de dialogue affichée si vous tentez de supprimer un dossier inexistant :



Vous voulez un code plus compact ? Essayez cette instruction :

```
RmDir "c:\data\excel"
```

Ou encore, en intégrant la gestion d'erreurs :

```
On Error GoTo gestionErreurs
```

```
RmDir "c:\data\excel"
```

```
End
```

```
gestionErreurs:
```

```
MsgBox "Erreur n° " & Err.Number & vbCrLf & Err.Description
```

Tester si un dossier existe

Pour tester si un dossier existe, commencez par créer un objet **Scripting.FileSystemObject** :

```
Dim fs As New Scripting.FileSystemObject
```

Vous pouvez alors utiliser la méthode **FolderExists** de cet objet en passant le nom du fichier en argument. Si le fichier existe, la méthode **FolderExists** retournera la valeur **True**. Dans le cas contraire, elle retournera la valeur **False**.

Pour terminer le code, supprimez l'objet **Scripting.FileSystemObject** de la mémoire en lui affectant la valeur **Nothing** :

```
Set fs = Nothing
```

Voici un exemple de code complet. Ici, si le dossier **c:\data\excel** n'existe pas, il est créé :

```
Dim fs As New Scripting.FileSystemObject  
  
If fs.FolderExists("c:\data\excel") = False Then  
  
    fs.CreateFolder "c:\data\excel"  
  
End If  
  
Set fs = Nothing
```

Copier un dossier dans un autre

Pour copier un dossier dans un autre, commencez par créer un objet **Scripting.FileSystemObject** :

```
Dim fs As New Scripting.FileSystemObject
```

Utilisez alors la méthode **CopyFolder** de cet objet en lui passant deux arguments : le chemin du dossier à copier et le chemin du dossier destination :

```
fs.CopyFolder "c:\data\excel", "c:\data\excel2"
```

Une fois la copie effectuée supprimer l'objet **Scripting.FileSystemObject** de la mémoire en lui affectant la valeur **Nothing** :

```
Set fs = Nothing
```

Voici un exemple de code complet. Ici, le dossier **c:\data\excel** est dupliqué dans le dossier **c:\data\excel2** :

```
Dim fs As New Scripting.FileSystemObject  
fs.CopyFolder "c:\data\excel",  
"c:\data\excel2"  
Set fs = Nothing
```

Remarque

Si un fichier de même nom est trouvé dans le dossier destination, il n'est pas écrasé. Y compris si sa taille est différente.

Déplacer un dossier dans un autre

Pour déplacer un dossier dans une autre dossier, trois étapes sont nécessaires :

- Création du dossier destination
- Copie du dossier source dans le dossier destination
- Suppression du dossier source

Ici par exemple, le dossier **c:\data\excel** est déplacé dans le dossier **e:\data\excel** :

```
Dim fs As New Scripting.FileSystemObject
```

```
fs.CreateFolder "e:\data\excel"
```

```

fs.CopyFolder "c:\data\excel", "e:\data\excel"

fs.DeleteFolder "c:\data\excel"

Set fs = Nothing

```

Attention

Pour que ce code fonctionne, les dossiers **c:\data\excel** et **e:\data** doivent exister. Après son exécution, les fichiers et dossiers contenus dans le dossier **c:\data\excel** se retrouvent dans le dossier **e:\data\excel**.

Modifier et lire les attributs des fichiers

Dans Windows, les fichiers possèdent des attributs qui peuvent changer leur comportement dans l'explorateur de fichiers. Par exemple, les fichiers cachés n'apparaîtront pas dans l'explorateur de fichiers, ou encore, les fichiers à lecture seule ne pourront pas être modifiés.

Modifier les attributs d'un fichier

En utilisant l'instruction VBA **SetAttr**, vous pouvez modifier l'attribut d'un fichier quelconque. Voici sa syntaxe :

```
SetAttr "chemin", attribut
```

Où **chemin** est le chemin complet du fichier dont vous voulez modifier l'attribut et **attribut** est l'attribut que vous voulez lui donner.

Les différents attributs utilisables sont résumés dans ce tableau :

Paramètre	Valeur numérique	Description
vbNormal	0	Fichier normal
vbReadOnly	1	Lecture seule
vbHidden	2	Fichier caché
vbSystem	4	Fichier système
vbArchive	32	Archive
vbAlias	64	Lien symbolique

Par exemple, pour affecter l'attribut Lecture seule au fichier **c:\data\a.pdf**, vous utiliserez cette instruction :

```
SetAttr "c:\data\a.pdf", vbReadOnly
```

Vous pouvez également passer la valeur numérique correspondante à **SetAttr** :

```
SetAttr "c:\data\a.pdf", 1
```

Si nécessaire, vous pouvez cumuler plusieurs attributs en les additionnant. Par exemple, pour affecter les attributs **Lecture seule** et **Fichier caché** au fichier **c:\data\a.pdf**, vous utiliserez cette instruction :

```
SetAttr "c:\data\ a.pdf", vbReadOnly + vbHidden Ou
```

encore :

```
SetAttr "c:\data\ a.pdf", 3
```

[Lire les attributs d'un fichier](#)

Vous voulez connaitre l'attribut d'un fichier ? Utilisez la fonction **GetAttr()** :

```
GetAttr("chemin")
```

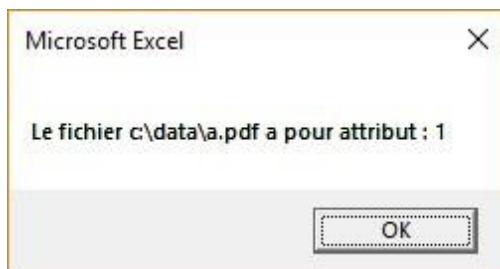
Où **chemin** est le chemin complet du fichier dont vous voulez connaitre les attributs.

Cette fonction retourne l'attribut du fichier spécifié sous une valeur numérique.

Par exemple, pour connaitre l'attribut de l'hypothétique fichier **c:\data\ a.pdf** et l'afficher dans une boîte de dialogue, vous utiliserez cette instruction :

```
MsgBox "Le fichier c:\data\ a.pdf a pour attribut : " &  
GetAttr("c:\data\ a.pdf")
```

Ici, le fichier est en lecture seule :



Accès aux dossiers spéciaux

Il est très simple de connaitre le chemin du dossier d'installation de Windows, du dossier système et du dossier des fichiers temporaires. Commencez par créer un objet **Scripting.FileSystemObject** :

```
Dim fs As New Scripting.FileSystemObject
```

Les dossiers spéciaux sont accessibles avec la fonction **GetSpecialFolder()** de l'objet **Scripting.FileSystemObject**.

Pour connaitre le dossier d'installation de Windows, passez la constante **WindowsFolder** ou la valeur **0** à cette fonction :

```
MsgBox fs.GetSpecialFolder(WindowsFolder)
```

```
MsgBox fs.GetSpecialFolder(0)
```

Pour connaitre le dossier système, passez la constante **SystemFolder** ou la valeur **1** à cette fonction :

```
MsgBox fs.GetSpecialFolder(SystemFolder)
```

```
MsgBox fs.GetSpecialFolder(1)
```

Pour connaitre le dossier des fichiers temporaires, passez la constante **TemporaryFolder** ou la valeur **2** à cette fonction :

```
MsgBox fs.GetSpecialFolder(TemporaryFolder)
```

```
MsgBox fs.GetSpecialFolder(2)
```

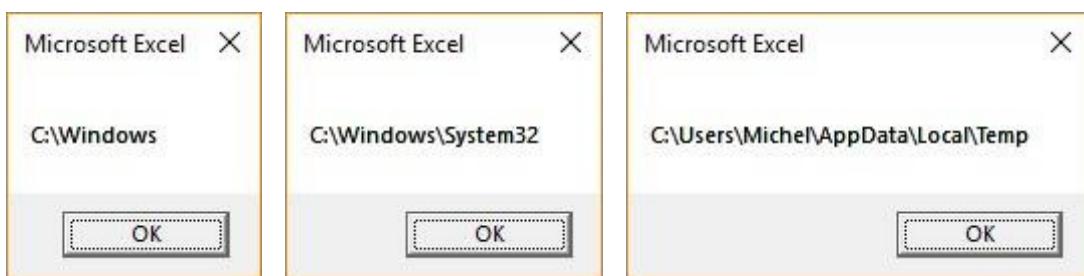
Une fois l'objet **Scripting.FileSystemObject** utilisé, supprimez-le de la mémoire en lui affectant la valeur **Nothing** :

```
Set fs = Nothing
```

Voici un exemple de code complet :

```
Dim fs As New Scripting.FileSystemObject  
  
MsgBox fs.GetSpecialFolder(WindowsFolder)  
  
MsgBox fs.GetSpecialFolder(SystemFolder)  
  
MsgBox fs.GetSpecialFolder(TemporaryFolder)  
  
Set fs = Nothing
```

Et voici le résultat :



Recherche de doublons dans une colonne

Cette section va vous présenter un code utile qui pourra certainement vous servir un jour ou l'autre. Le but du jeu est de trouver les cellules qui apparaissent en double (ou plus) dans la colonne où se trouve la cellule active. Il n'y a rien de bien compliqué : il suffit de comparer tour à tour toutes les cellules de la colonne à la première, puis à la deuxième, puis à la troisième, ainsi de suite jusqu'à l'avant-dernière cellule de la colonne. Ici, la couleur d'arrière-plan de la cellule qui apparaît en double sera modifiée. Mais rien ne vous empêche de modifier le code pour faire tout autre chose si vous le souhaitez. Voici le code utilisé :

```
Sub RechercherDoublons()

    Dim col, nbCells, i, j

    col = ActiveCell.Column

    nbCells = Application.WorksheetFunction.CountA(Range(Columns(col),
    Columns(col)))

    For i = 1 To nbCells - 1

        For j = i + 1 To nbCells

            If Cells(i, col) = Cells(j, col) Then

                Cells(j, col).Interior.Color = RGB(255, 0, 0)

            End If

        Next j

    Next i

End Sub
```

Examinons ce code.

La première ligne définit les variables qui seront utilisées dans le code :

```
Dim col, nbCells, i, j
```

La deuxième ligne stocke dans la variable **col** la colonne dans laquelle se trouve la cellule active. C'est dans cette colonne que les doublons seront recherchés :

```
col = ActiveCell.Column
```

La troisième ligne compte le nombre de cellules non vides de la colonne col. Le résultat est stocké dans la variable **nbCells** :

```
nbCells = Application.WorksheetFunction.CountA(Range(Columns(col),  
Columns(col)))
```

Le bloc de code suivant utilise deux boucles **For Next** imbriquées. La première parcourt les cellules de la première (**1**) à l'avant-dernière (**nbCells-1**) :

```
For i = 1 To nbCells - 1
```

La seconde parcourt les cellules d'indices compris entre **i+1** et **nbCells** :

```
For j = i + 1 To nbCells
```

Si les cellules d'indice **i, col** et **j, col** sont identiques, un doublon a été trouvé :

```
If Cells(i, col) = Cells(j, col) Then
```

Dans ce cas, la cellule d'indice **j, col** est colorée en rouge :

```
Cells(j, col).Interior.Color = RGB(255, 0, 0)
```

Exercices

- Exercices sur les bases de VBA

1. **Calcul de pourcentage :**

Créez une procédure qui demande à l'utilisateur d'entrer un montant et un pourcentage, puis calcule et affiche la valeur du pourcentage appliqué.

Exemple : Pour un montant de 1000 et un pourcentage de 5 %, le programme doit afficher 50.

2. **Conversion devises :**

Créez une procédure qui convertit un montant saisi par l'utilisateur d'euros en dollars, en utilisant un taux de change défini dans une cellule Excel.

- **Bonus :** Gérer les erreurs si le taux de change n'est pas défini.

3. **Analyse simple des notes :**

Écrivez une fonction qui :

- Prend une note entre 0 et 20 en paramètre.
- Retourne "Excellent" pour une note supérieure à 16, "Bon" entre 10 et 16, et "Insuffisant" pour une note inférieure à 10.

- Exercices sur les boucles et tableaux

4. **Somme des valeurs :**

Créez une procédure qui :

- Parcourt les cellules d'une plage (par exemple A1:A10).
- Calcule la somme des valeurs numériques dans cette plage et l'affiche dans une cellule ou un message.

5. **Statistiques financières :**

Créez une procédure qui :

- Parcourt une liste de prix d'actions dans une colonne.
- Calcule le prix moyen, le prix minimum, et le prix maximum.
- Affiche les résultats dans des cellules Excel spécifiques.

6. **Filtrage dynamique :**

- Créez une procédure qui copie toutes les transactions financières supérieures à 1 000 € d'une colonne vers une autre colonne.
- **Bonus :** Utilisez un tableau dynamique pour stocker temporairement les données.

- Exercices sur la manipulation d'Excel

7. **Lecture de données dynamiques :**

Écrivez une procédure qui :

- Lit les valeurs d'une plage de cellules contenant des montants (par exemple B2:B20).
- Compte le nombre de montants supérieurs à 5 000.
- Affiche ce nombre dans une cellule ou un message.

8. Fusionner des feuilles :

- Créez une procédure qui :
 1. Parcourt toutes les feuilles d'un classeur.
 2. Copie les données de chaque feuille dans une feuille unique appelée "Synthèse".
- **Bonus :** Ajouter une colonne indiquant le nom de la feuille d'origine.

9. Création de graphiques dynamiques :

- Créez une procédure qui génère automatiquement un graphique (par exemple un histogramme ou une courbe) basé sur des données dans une plage définie (par exemple A1:B10).
- **Bonus :** Ajouter un titre personnalisé au graphique.

● **Exercices sur la gestion des fichiers**

10. Importer un fichier CSV :

- Créez une procédure qui demande à l'utilisateur de sélectionner un fichier CSV, puis importe son contenu dans une feuille Excel.

11. Exporter vers un fichier texte :

- Créez une procédure qui parcourt une plage de cellules Excel et exporte chaque ligne dans un fichier texte.
- **Bonus :** Ajoutez une option pour choisir le séparateur (ex. : virgule ou point-virgule).

12. Rapport automatisé :

- Créez une macro qui :
 1. Lit les données d'une feuille Excel (par exemple des ventes par région).
 2. Génère un rapport résumé dans une nouvelle feuille (par exemple total par région).
 3. Sauvegarde automatiquement le rapport au format PDF.

● **Exercice final : Automatisation avancée**

13. Tableau de bord financier automatisé :

- Créez une macro complète qui :
 1. Lit des données financières brutes (par exemple, des transactions ou des revenus).
 2. Effectue des calculs (totaux, moyennes, écarts).
 3. Génère un tableau de bord avec des graphiques et des mises en forme conditionnelles.
 4. Sauvegarde automatiquement le fichier final.

Prochaine Étape

Pour aller plus loin et approfondir vos compétences, nous vous invitons à consulter le compte [GitHub](#) dédié à ce cours. Vous y trouverez un récapitulatif complet des notions abordées ainsi que des **exercices pratiques** pour renforcer votre apprentissage. Les exercices vous permettront de mettre en pratique les concepts étudiés et de vous familiariser davantage avec VBA.

N'hésitez pas à consulter régulièrement le dépôt GitHub pour des mises à jour et de nouveaux contenus !

Accédez aux fichiers exercices ici : <https://github.com/Evilafo/Cours-VBA>

Bon courage pour la suite et bonne pratique de VBA !