

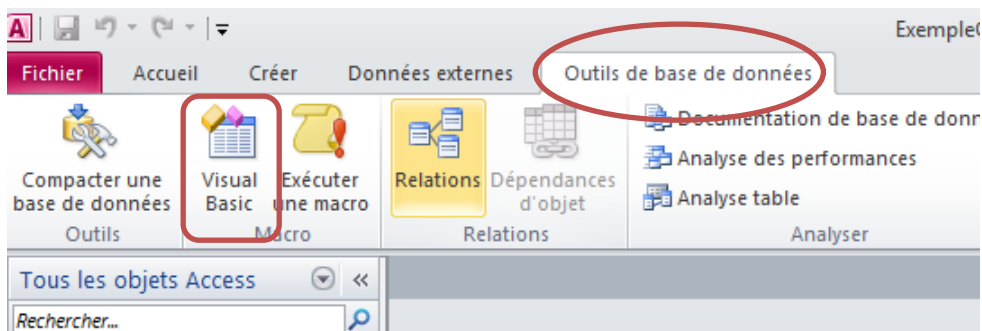
Fiche de TD VBA

L'objectif de cette fiche est de faire un rappel (voire une présentation rapide) du langage de programmation VBA et de son usage sur des documents Excel et Access. Pour rappel, VBA (*Visual Basic for Applications*) est à la fois un langage de programmation et un environnement d'exécution attachés aux documents MS Office (Word, Excel, Access...). Il nous permet d'enrichir les documents Office avec un comportement dit actif, tels que des vérifications ou l'automatisation d'un traitement. Les documents deviennent ainsi plus dynamiques, pour devenir presque des véritables applications.

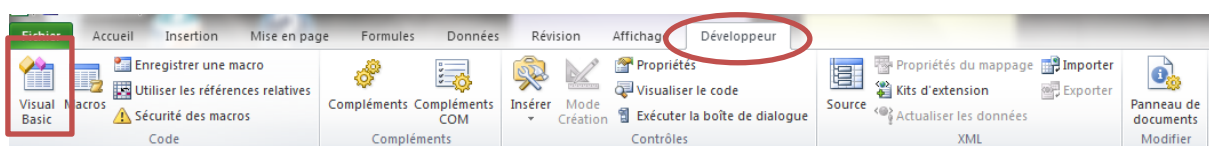
Démarrage

Pour commencer, il faut d'abord comprendre en quoi VBA peut nous être utile. VBA se positionne dans la suite logique des « macros ». Les macros, qui sont prises en charge par les documents aux formats **.xlsm**, **.accdb** ou encore **.docm**, représentent une suite d'opérations réalisées de manière séquentielle. Lorsqu'on exécute une macro, celle-ci reproduit un ensemble d'actions enregistrées préalablement. Or, l'enregistreur de macro ne peut enregistrer que des suites d'actions (qu'il traduit d'ailleurs en VBA). Dès qu'il faut faire un traitement un peu plus complexe (des vérifications sur un ensemble de données, par exemple), il faut passer au VBA, car les macros ne pourront pas enregistrer tous les traitements.

Pour faire de VBA, il faut avoir accès à son **environnement de programmation**, nommé VBE, qui est intégré aux applications MS Office. Dans Access, c'est facile, le bouton d'accès à l'environnement VBE est déjà disponible sur l'onglet « Outils de base de données », illustré ci-dessous.

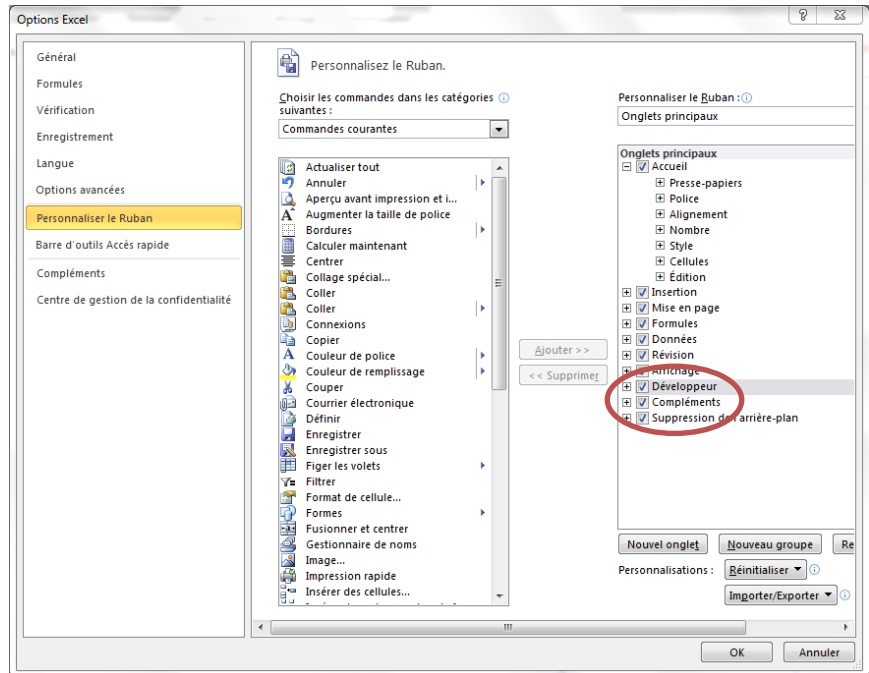


Sur Excel, il faut d'abord rendre visible l'onglet « **Développeur** » sur le ruban, pour qu'on y trouve le bouton d'accès à l'environnement (« Visual Basic »).

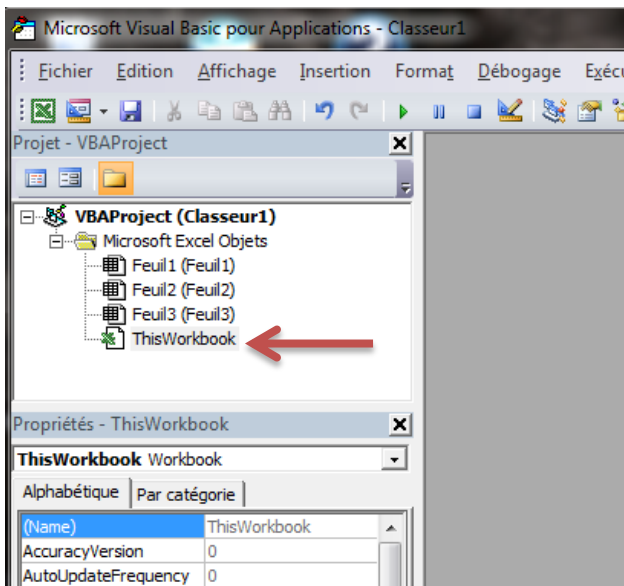


Activité

Ouvrir Excel, aller sur les « Options » → « Personnaliser le Ruban » et cocher la case « Développeur », comme l'illustre la figure ci-contre :



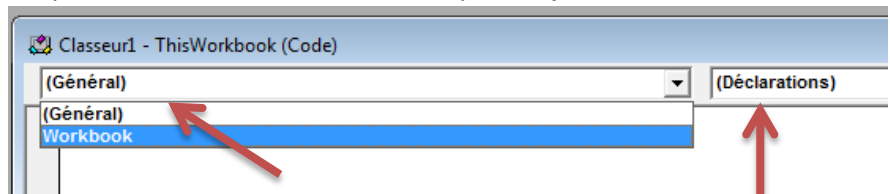
Une fois visible l'onglet « Développeur », ouvrir l'environnement VBE en cliquant sur le bouton « **Visual Basic** ». L'environnement ci-dessous devient alors visible.



Maintenant, nous allons créer notre premier « code » VBA. Nous allons programmer un petit message de **bienvenue** qui s'affichera lorsqu'on **ouvre le fichier Excel**. Pour ce faire, il nous faudra donc programmer **l'événement « Open »** de notre classeur.

Dans l'environnement VBE, faire un double click sur le module « **ThisWorkbook** ». Dans la fenêtre que s'ouvre, sélectionner « **Workbook** » dans le premier menu déroulant, et « **Open** » sur le second (normalement, Excel fera cela pour vous). Dans ce deuxième menu, vous trouverez tous les **événements** auxquels peut répondre l'objet **Workbook**.

Une fois sélectionné l'événement « Open », l'environnement vous remplira déjà le début de votre code, avec la déclaration de la procédure (**Private Sub Workbook_Open() ... End Sub**). Il nous reste qu'à la remplir. 😊



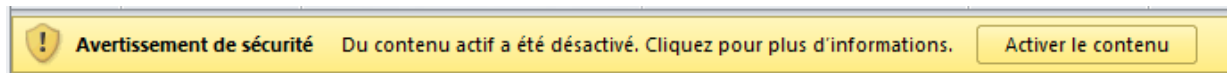
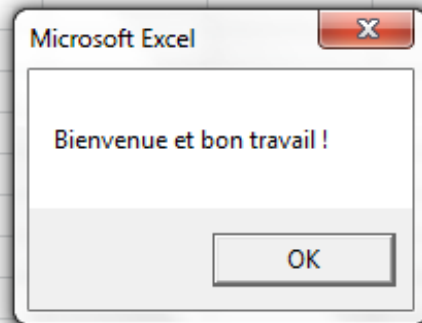
Pour ce premier exemple, nous allons utiliser une fonction offerte par VBA appelée **MsgBox**. Celle-ci permet d'afficher des fenêtres de dialogues de différents types (simples, avec deux boutons OK et Cancel, etc.), en fonction des paramètres qu'on lui donne. On l'utilisera dans sa version la plus simple, avec juste un

message à afficher. Nous allons donc taper la ligne ci-contre dans notre **procédure Sub** comme l'illustre la figure ci-dessous :

```

Classeur1 - ThisWorkbook (Code)
Workbook [v] Open
Private Sub Workbook_Open()
  MsgBox "Bienvenue et bon travail ! "
End Sub
  
```

Il nous reste qu'à **enregistrer** notre document (**attention au format : il doit être au format .xlsm**), **refermer** l'environnement VBA et le classeur, et le **rouvrir** pour voir si notre événement va bien marcher (s'il va bien se déclencher à l'ouverture du classeur). Lors de l'ouverture du document, Excel nous demandera si on veut « **activer le contenu** » (figure ci-dessous), c'est-à-dire, autoriser les macros et le contenu VBA du document (à faire uniquement avec les documents dont on connaît la provenance). Une fois autorisé le contenu, nous devons voir l'exécution de notre événement « Open ».



Exercices VBA sur Excel

1) Améliorer l'exercice précédent, en ajoutant au message affiché le nom du document ouvert.

Pour réaliser cet exercice, nous allons manipuler l'objet **ThisWorkbook**, qui représente le classeur ouvert, et plus précisément, sa propriété « **Name** » : **ThisWorkbook.Name**. Utiliser les astuces de concaténation et de nouvelle ligne (voir section « *Bon à savoir* » en fin du document) pour ajouter le nom du document au message. Utiliser le bouton « exécuter » pour tester votre code.

```

activite1.xlsm - ThisWorkbook (Code)
Workbook [v] Open
Private Sub Workbook_Open()
  MsgBox "Bienvenue et bon travail sur le fichier "
    & ThisWorkbook.Name
End Sub
  
```



```

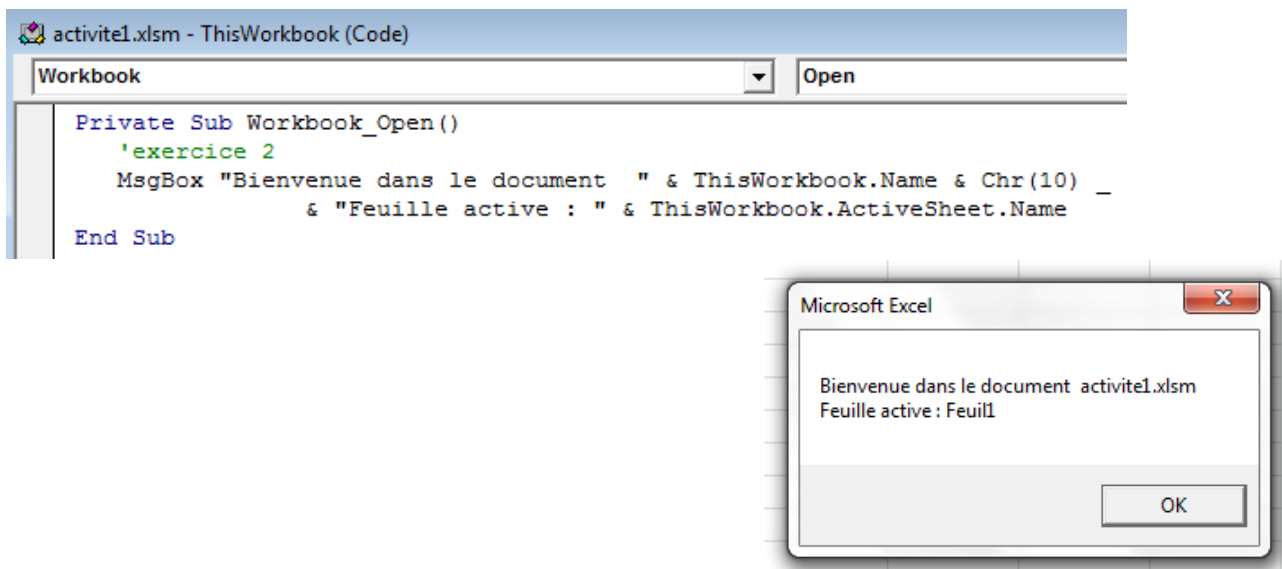
Private Sub Workbook_Open()

  MsgBox "Bienvenue et bon travail ! "

End Sub
  
```

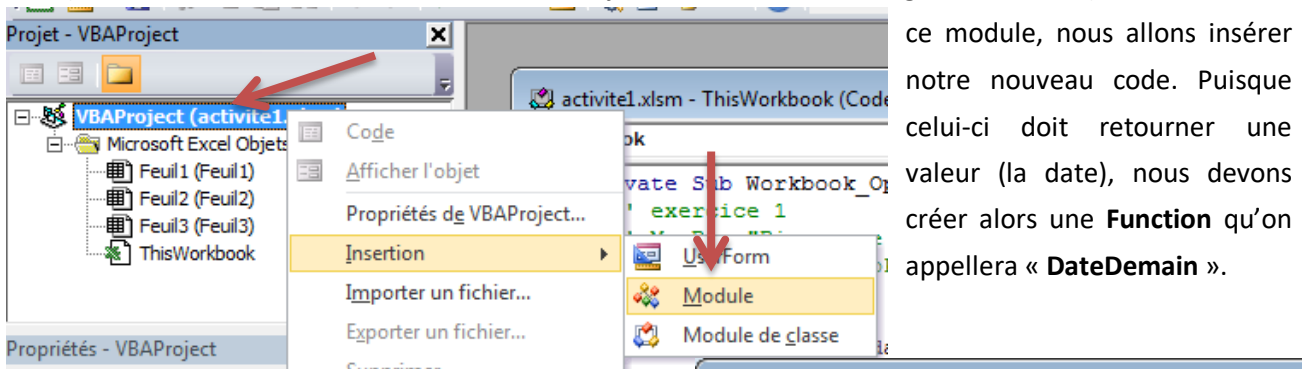
2) Améliorer encore plus le code précédent, en ajoutant au message le nom de feuille active.

Lorsqu'on ouvre un classeur, la feuille active est accessible à travers l'objet **ActiveSheet**, appartenant à **ThisWorkbook**. Nous allons, à nouveau, utiliser la propriété **Name**, cette fois pour l'**ActiveSheet**. Tester votre code à l'aide du bouton « exécuter ».



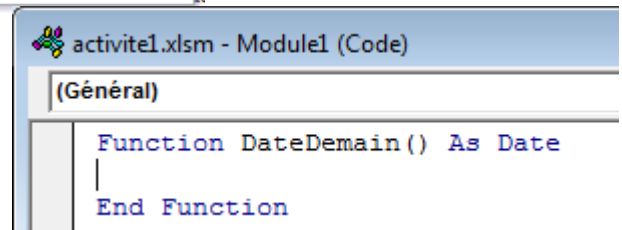
3) Construire une fonction qui retourne la date de demain.

Afin de construire notre première fonction, nous allons d'abord ajouter un **nouveau module** à notre projet VBA (menu « **insertion** » ou click droit sur VBAProject comme l'illustre la figure ci-dessous). A l'intérieur de



ce module, nous allons insérer notre nouveau code. Puisque celui-ci doit retourner une valeur (la date), nous devons créer alors une **Function** qu'on appellera « **DateDemain** ».

Afin d'obtenir la date de demain, il faut d'abord trouver celle d'aujourd'hui. Pour cela, nous allons faire appel à la fonction « **Date** » de VBA, dont la valeur nous garderons dans une variable nommée « **auj** » de type



« **Date** » (comme nous avons vu en cours). La date de demain devient donc « **auj + 1** ». Sachant que la valeur que retournera une fonction correspond à la valeur de la variable de même nom que cette fonction, il nous reste donc à indiquer : « **DateDemain = auj + 1** » (voir figure ci-dessous).

activite1.xlsm - Module1 (Code)

(Général)

```
Function DateDemain() As Date
  Dim auj As Date
  auj = Date
  DateDemain = auj + 1
End Function
```

Avant

f_x	=DateDemain()
C	D
	41689

Après l'avoir ajusté →

← Avant d'ajuster le format de la cellule

f_x	=DateDemain()
C	D
	19/02/2014

Maintenant que nous avons notre fonction, il suffit de l'utiliser dans notre classeur. Ajouter donc la formule « **=DateDemain()** » dans une cellule du classeur pour utiliser votre fonction. **Attention** : n'oubliez pas de **formater votre cellule** en tant que **date**, sinon la cellule présentera la date sous sa forme « numérique » (c'est-à-dire, le nombre de jours écoulés depuis 1/1/1970).

4) Construire une fonction qui vérifie si une année (passée en paramètre) est bissextile ou non.

Pour faire cet exercice, nous allons d'abord remplir une colonne de notre classeur avec des années : 2000, 2001, 2002... 2015. Puis, nous allons construire notre fonction, qu'on nommera « **EstBissextile** ». L'idée ici est de recevoir en **paramètre** un numéro correspondant à une année (**annee As Integer**) et donc de répondre si oui ou non cette année est bissextile (retour « **As Boolean** »).

```
End Function

Function EstBissextile(annee As Integer) As Boolean
End Function
```

En principe, on dit qu'une année est bissextile si elle est divisible par 4. En d'autres termes, si le reste de la division de l'année par 4 est zéro, l'année est bissextile. Sinon, elle ne l'est pas. Nous avons donc deux étapes à réaliser : d'abord récupérer le reste de la division, puis vérifier si celui-ci est égale à zéro ou non. Nous pouvons réaliser la première étape grâce à l'opérateur « **Mod** » : l'opération « **annee Mod 4** » nous donnera comme résultat le reste de la division, qu'on pourra stocker dans une variable.

```
Dim reste As Integer
reste = annee Mod 4
```

Pour tester si le reste est égal à 0, il nous faut réaliser une instruction conditionnelle « **If... then... else** », que nous avons vu en cours. Enfin, comme pour l'exercice précédente, le résultat de la fonction (**True** si l'année est bissextile, **False** sinon) doit être gardé dans une variable du même nom de la fonction (**EstBissextile** dans notre cas).

Maintenant que notre fonction est prête, on va l'utiliser pour vérifier les valeurs des années sur notre classeur.

```
Function EstBissextile(annee As Integer) As Boolean
  Dim reste As Integer
  reste = annee Mod 4
  If reste = 0 Then
    EstBissextile = True
  Else
    EstBissextile = False
  End If
End Function
```

Police		Alignem	
X	✓	f _x	=EstBissextile(C2)
		C	D
		2000	=EstBissextile(C2)
		2001	
		2002	

f _x	=EstBissextile(C3)	
C	D	
2000	VRAI	
2001	FAUX	
2002	FAUX	
2003	FAUX	
2004	VRAI	
2005	FAUX	

5) Améliorer l'exercice précédent pour prendre en considération les exceptions: les années divisibles par 100 et pas par 400.

L'algorithme que nous avons utilisé dans l'exercice précédent comporte une faille : les années divisibles par 4 et par 100 ne sont pas tous bissextiles. Par exemple, l'année 1900 ou l'année 2100 : toutes les deux sont divisibles par 4 ($1900 \div 4 = 475$) et par 100 ($1900 \div 100 = 19$), mais elles ne sont pas bissextiles. Pour qu'une année divisible par 4 et par 100 soit bissextile, elle doit être aussi divisible par 400. Nous devons donc ajouter des tests supplémentaires, comme l'illustre le diagramme d'activités ci-contre. *A vous de jouer ! ☺*

Function EstBissextile(annee As Integer) As Boolean

'exercice 4

Dim reste As Integer

reste = annee Mod 4

If reste = 0 Then

EstBissextile = True

'exercice 5

If annee Mod 100 = 0 Then

If annee Mod 400 = 0 Then

EstBissextile = True

Else

EstBissextile = False

End If

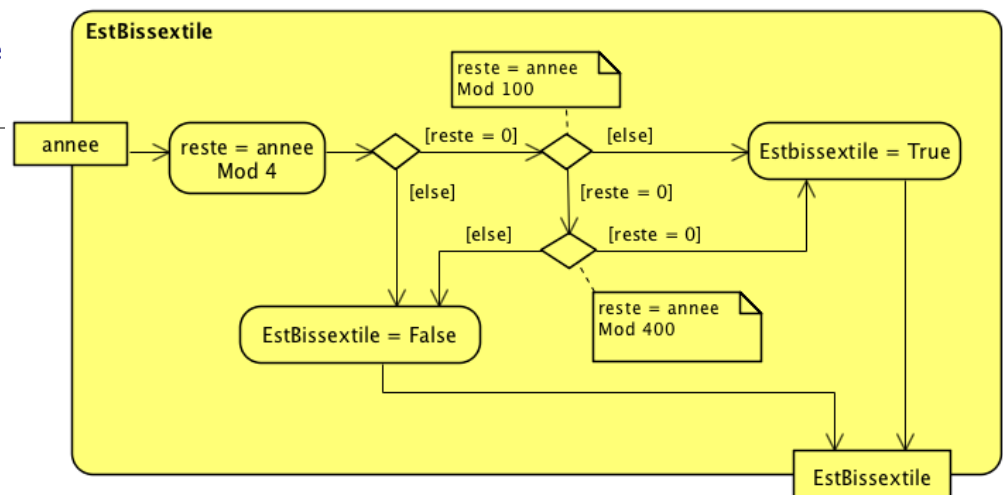
End If

Else

EstBissextile = False

End If

End Function



6) Créer une fonction capable de compter combien d'années bissextiles existent dans un ensemble de cellules contenant des années.

L'objectif de cet exercice est de pouvoir parcourir un ensemble de cellules contenant des années et de compter combien, parmi ces années, sont bissextiles (à l'aide de notre fonction *EstBissextile*). Pour le faire, nous aurons besoin de connaître l'ensemble de cellules à parcourir. Il nous faut donc un objet **Range** en paramètre, lequel va nous indiquer quelles cellules nous allons regarder.

```
'exercice 6
Function SommeBissextile(unRange As Range) As Integer
    'on va compter le nombre d'annees bissextiles qui sont dans la rangee d
    Dim somme As Integer
    |
    End Function
```

← un compteur, pour compter combien de fois on a trouvé une année bissextile

Ensuite, il va falloir parcourir chacune de ces cellules à l'aide d'une boucle. Dans VBA, les boucles de type « **For Each ... Next** », vues en cours, permettent de parcourir un à un les éléments appartenant à un ensemble (une collection, par exemple). On va donc pouvoir l'utiliser pour regarder une par une de nos cellules.

For Each cel In unRange
Next

Pour chacune des cellules, on va devoir vérifier, à l'aide d'un « **If ... Then** », si la valeur de la cellule est Bissextile ou non, à l'aide de la fonction que nous avons fait précédemment. Si c'est bien le cas, on incrémente notre compteur (**somme = somme + 1**). Une fois terminée la boucle, le compteur contiendra combien de fois on a trouvé une année bissextile. Nous n'avons qu'à retourner la valeur enregistrée sur le compteur (**SommeBissextile = somme**).

```
'exercice 6
Function SommeBissextile(unRange As Range) As Integer
    'on va compter le nombre d'annees bissextiles qui sont dans la rangee
    Dim somme As Integer
    Dim cel As Range

    'pour chaque cellule dans le range
    For Each cel In unRange
        'si elle est bissextile, on la somme
        If EstBissextile(cel.Value) Then
            somme = somme + 1
        End If
    Next

    SommeBissextile = somme

End Function
```

Puis, il nous reste que tester notre nouvelle fonction sur une rangée de cellules de notre tableur.

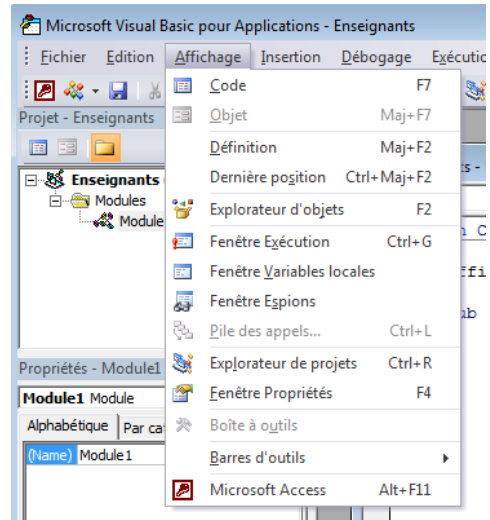
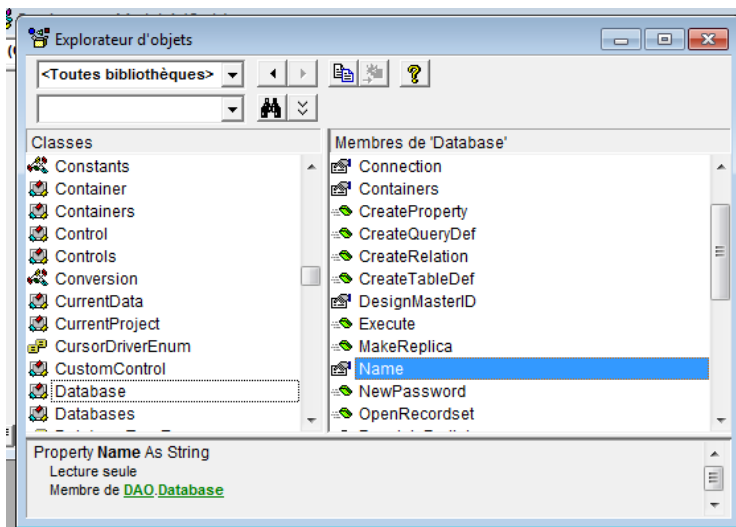
Compléments		Contrôle	
fx =SommeBissextile(C2:C6)			
C	D	E	
le(C2:C6)	19/02/2014		
2000	VRAI		
2001	FAUX		
2002	FAUX		
2003	FAUX		
2004	VRAI		

fx =SommeBissextile(C2:C6)			
C	D	E	
2	19/02/2014		
2000	VRAI		
2001	FAUX		
2002	FAUX		
2003	FAUX		
2004	VRAI		

Bon à savoir

Explorateur d'objets

Lorsqu'on veut trouver un objet et savoir ses propriétés et opérations, on peut utiliser l'explorateur d'objets. Celui-ci permet de rechercher un objet ou classe par son nom et indique les propriétés et opérations disponibles pour un objet sélectionné.



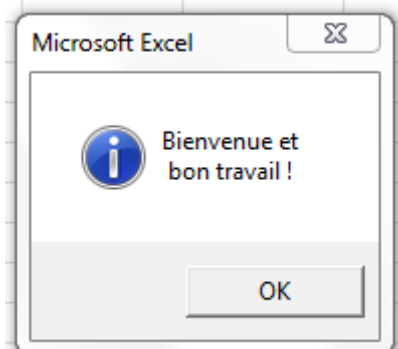
Concaténation et nouvelle ligne

Toutes les commandes VBA doivent entrer dans une seule ligne, ce qui n'est pas très pratique lorsqu'on a une ligne de code trop

longue. Afin de pouvoir « casser » la commande en plusieurs lignes, on va utiliser le caractère « _ ».

Inversement, si on souhaite regrouper (concaténer) plusieurs chaînes de caractères dans une seule (par exemple, dans un message pour un MsgBox), on va pouvoir utiliser le caractère « & », comme ici :

```
Private Sub Workbook_Open()
    MsgBox "Bienvenue et bon travail ! " _
        , vbInformation
End Sub
```



```
Private Sub Workbook_Open()
    MsgBox "Bienvenue et " & " bon travail ! "
End Sub
```

on peut utiliser la fonction « Chr(10) » :

Enfin, si on souhaite ajouter à une chaîne de caractère une nouvelle ligne pour qu'elle soit affichée (dans un MsgBox, par exemple),

```
Private Sub Workbook_Open()
    MsgBox & Chr(10) _
        & " bon travail ! " _
        , vbInformation
End Sub
```

Déclaration d'une variable

Afin de déclarer une variable dans un code VBA, il suffit d'utiliser le mot clé « **Dim** ». On peut lui associer, à l'aide du mot clé « **As** », un type de donnée (*String*, *Date*, *Integer*...) ou une classe (*Range*, *RecordSet*...). Dans ce dernier cas, notre variable garde, en réalité, un objet.

Une fois, la variable déclarée, on peut lui attribuer une valeur à l'aide de l'opérateur « = ». **Attention**, par contre, s'il s'agit d'un objet, l'attribution se fait à l'aide du mot clé « **Set** ».

```
Dim i As Integer
Dim cellule As Range
```

```
i = 1
Set cellule = Worksheets("Feuil1").Cells(1, 1)
```

Obtenir le contenu d'une cellule

On peut accéder à n'importe quelle cellule dans un classeur à l'aide des collections. La collection **Worksheets** permet de retrouver les feuilles de calcul par leur nom (ou position, à compter à partir de 1), alors que l'objet **ActiveSheet** nous offre la feuille active.

A partir d'un objet **Worksheet**, on peut récupérer n'importe quelle cellule (ou plage de cellules), grâce à la collection **Cells**, ou encore l'opération **range()**. Une cellule (ou plage de cellule) va donc être représentée par un objet **Range**, pour lequel on peut solliciter sa valeur avec la propriété **Value**, ou encore modifier ses propriétés graphiques.

```
Dim feuille As Worksheet
Set feuille = Worksheets("Feuil1")
```

```
Dim cellule As Range
Set cellule = feuille.Cells("A1")
Set cellule = ActiveSheet.Cells(1,1)
```

```
MsgBox cellule.Value
cellule.Font.Bold = True
cellule.Font.Size = 14
```

```
Dim cellule As Range
Set cellule = ActiveSheet.range("A1 :B1")
```