Chapitre 5
Les objets et collections en VBA

1. Notion d'objet

Le VBA est un langage qui permet de faire de la programmation orientée objet (POO) : un objet représente une idée, un concept ou toute entité du monde réel, comme un avion, un individu ou encore un film. Il possède une structure interne et un comportement, et peut communiquer avec ses pairs. Les éléments qui permettent de décrire un objet forment ce qu'on appelle une classe. Chaque objet issu d'une classe est une instance de classe. Les classes comportent des propriétés, des méthodes et des événements.

1.1 Propriétés

L'objet est une entité que l'on peut distinguer grâce à ses propriétés (sa couleur, ses dimensions, par exemple). Si l'on prend par exemple un livre, il est caractérisé par ses propriétés : nombre de pages, titre, nombre de chapitres, éditeur, contenu, etc. Chacune de ses propriétés peut être spécifique à chaque livre, mais tous les livres possèdent fondamentalement les mêmes propriétés. Certaines propriétés des objets peuvent être modifiées (vitesse d'une voiture par exemple) et d'autres non (une marque de voiture).

En programmation VBA, la syntaxe générale d'accès aux propriétés d'un objet est la suivante :

```
UnObjet.SaPropriete
```

On accède ici à la propriété `SaPropriete` de l'objet `UnObjet`. La combinaison `MonObjet.SaPropriete` aura le même comportement qu'une variable classique, pouvant prendre une valeur ou retourner une valeur avec l'utilisation de l'opérateur =.

Par exemple, il est possible de lire le contenu SQL d'une requête Access et de l'afficher avec le code suivant :

```
Sub LireSQLRequete()
    Dim UneRequete As QueryDef
    UneRequete.SQL = "SELECT * FROM MaTable"
    ...
    MsgBox UneRequete.SQL
End Sub
```

Nous reviendrons plus en détail sur l'objet `QueryDef` dans le chapitre Les objets d'accès aux données DAO et ADO.

Une propriété d'un objet peut être elle-même un objet, ce qui pourra être utilisé par la suite dans le code.

1.2 Méthodes

En plus des éléments qui caractérisent un objet, il est également possible de réaliser des actions avec ces objets, comme par exemple "asseoir", "se coucher", "faire le beau", etc. Ces actions sont appelées des méthodes. Ces actions sont représentées sous la forme de procédures et de fonctions en VBA selon qu'elles peuvent ou non retourner des valeurs. Certaines actions peuvent nécessiter des paramètres pour fonctionner (nombre de caractères dans un livre, avec ou sans espace).

En programmation VBA, la syntaxe générale d'accès aux méthodes d'un objet est la suivante :

```
UnObjet.SaMethode [Monparametre]
```

Dans l'exemple suivant, il s'agit de simplement rafraîchir le lien entre une table liée et l'application Access :

```
Sub RafraichirLienTable()
    Dim UneVariableTemporaire As TableDef
    ...
    UneVariableTemporaire.RefreshLink
End Sub
```

Et dans le cas où la méthode est une fonction, on peut stocker le résultat dans une variable comme dans l'exemple suivant :

```
Sub Nombre_Champs()
    Dim Nb_Champs As Integer
    Dim MaTable As TableDef
    ...
Nb = MaTable.Fields.Count
End Sub
```

De la même façon que dans les appels de fonctions ou de procédures, les paramètres sont séparés par des virgules dans les appels de méthodes d'objet.

1.3 Événements

Pour chaque objet, il est possible de détecter et de prendre en considération le résultat d'actions particulières externes, que l'on appelle événements. Tous les événements qui ont lieu au cours de l'exécution d'un programme sont détectés et gérés par la machine, mais selon le choix du développeur, seuls certains peuvent faire l'objet d'un traitement particulier, comme par exemple un clic sur un bouton, une case que l'on coche, une ouverture ou fermeture de fenêtre, etc. Ces événements sont gérés au travers de procédures ayant tantôt des paramètres, tantôt non.

En VBA, la syntaxe la plus fréquente des événements est la suivante :

```
Sub MonObjet_MonEvenementDetecte()
    'Le code qui s'exécutera lorsque l'événement sera détecté
End Sub
```

Par exemple, lorsqu'il s'agit d'un clic sur un bouton `MonBouton`, on aura le code suivant :

```
Sub MonBouton_Click()
    'code qui s'exécutera
End Sub
```

Les événements sous Access seront traités plus en détail dans le chapitre Les événements Access.

1.4 Les collections

Lorsque plusieurs objets d'une même classe sont regroupés, ils peuvent appartenir à une même collection d'objets. Pour se référer à un élément en particulier d'une collection d'objets, plusieurs syntaxes sont possibles, parmi les suivantes :

```
Nom_Collection!Nom_Objet
Nom_Collection![nomObjet]
Nom_Collection("NomObjet")
Nom_Collection(variable_Nom) 'variable_Nom est une chaîne de
caractères contenant le nom de l'objet
Nom_Collection(variable_Numero) 'variable_Numero est une valeur
numérique contenant le numéro de l'objet au sein de la collection.
```

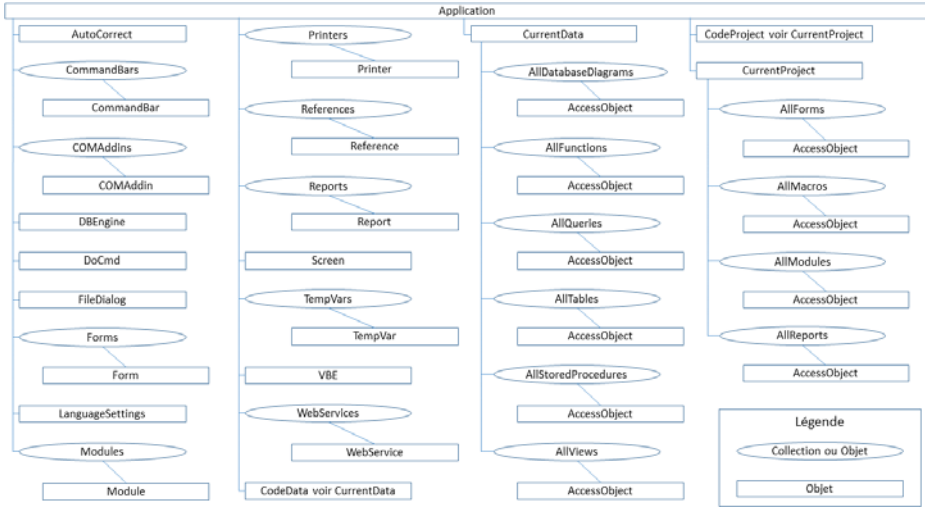
Les deux syntaxes les plus couramment utilisées étant les troisième et cinquième, car elles permettent l'usage de l'IntelliSense (voir chapitre VBE et la sécurité Access 2019).

Remarque

Il est d'usage dans les collections de commencer le décompte à 0, et non à 1. De plus, le numéro d'un objet au sein d'une collection dépendra de la présence d'autres éléments avant ou après lui, ce qui n'assure pas la certitude de tomber sur le bon objet par ce biais.

2. Modèle objet Access

L'objectif des quelques sections qui vont suivre est de montrer le modèle de hiérarchie utilisé au sein de l'application Access, sous forme de collections d'objets, qui vont être expliqués par la suite.



3. Collections Access

Voici les principales collections que l'on peut manipuler en VBA sous Access.

Collection	Contient une collection de	Description
`COMAddins`	`COMAddin`	Collection des **compléments COM**
`CommandBars`	`CommandBar`	Collection des **barres de commande**
`Forms`	`Form`	Collection des **formulaires ouverts**. Voir également `CurrentProject.AllForms`.

Collection	Contient une collection de	Description
`Modules`	`Module`	Collection des **modules**
`Printers`	`Printer`	Collection des **imprimantes disponibles**
`References`	`Reference`	Collection des **références bibliothèques**. Voir Menu : **Outils\Références**
`Reports`	`Report`	Collection des **états**. Voir également `CurrentProject.AllReports`
`TempVars`	`TempVar`	Collection des **variables temporaires**
`WebServices`	`WebService`	Collection des connexions à des **services web**

4. Objets Access

Voici les principaux objets qu'il est possible de manipuler dans le modèle Access.

Objet	Description
`Application`	Représente l'application Microsoft Access active.
`AutoCorrect`	Représente les options de correction automatique d'Access.
`DBEngine`	Représente le moteur de base de données Microsoft Jet. Cet objet permet de contrôler tous les autres objets d'accès aux données.
`DoCmd`	Permet de convertir en VBA des actions Macros.
`FileDialog`	Permet d'accéder aux fonctionnalités des boîtes de dialogue (Ouvrir ou Enregistrer par exemple).

Chapitre 2
Variables - Constantes - Types de données

Durée : 1 heure 10

Mots-clés

déclaration, portée, durée de vie, type, affectation, argument, type de données VBA, type de données utilisateur, membre, conversion, variable de type objet

Objectifs

Maîtriser l'emploi des variables et des constantes pour l'écriture des procédures et la réalisation de programmes.

Prérequis

Pour valider les prérequis nécessaires, avant d'aborder le TP, répondez aux questions ci-après (certaines questions peuvent nécessiter plusieurs réponses) :

1. La déclaration des variables dans VBA :
   - **a.** est réalisée avec l'instruction `Option Explicit` dans la partie déclaration du module.
   - **b.** peut être étendue à l'ensemble des modules.
   - **c.** est obligatoire.
   - **d.** doit être suivie obligatoirement du type de données.
2. Un nom de variable :
   - **a.** peut contenir un espace.
   - **b.** doit commencer par une lettre.
   - **c.** doit être unique au sein d'une même portée.
3. Une variable est accessible uniquement par les procédures de son module quand elle est déclarée avec le mot-clé :
   - **a.** `Dim`, au sein de la procédure.
   - **b.** `Dim`, dans la partie déclaration du module.
   - **c.** `Private`.
   - **d.** `Public`.

4. Lorsqu'une variable perd sa portée :
   - **a.** elle devient accessible à toutes les procédures.
   - **b** elle n'a plus de valeur sauf si elle est déclarée statique.
   - **c** elle perd son type.
5. Les déclarations de variables suivantes sont correctes dans la partie déclaration du module :
   - **a.** `Public varTest1`
   - **b.** `Private dblTest2 As Double`
   - **c.** `dblTest3 As Double`
6. Dans l'instruction suivante `Dim strMot, strPhrase as String`, la variable `strMot` est de type :
   - **a.** `String`
   - **b.** `Variant`
   - **c.** `inconnu`
7. Pour déclarer la constante publique `Pi`, on écrit :
   - **a.** `Public Const Pi As Double = 3.1415926`
   - **b.** `Const Pi Public = 3.1415926 As Double`
   - **c.** `Const Pi = 3.1415926`
8. Le type de données par défaut des variables est :
   - **a.** `Byte`
   - **b.** `String`
   - **c.** `Variant`
9. La déclaration du type s'effectue avec le mot-clé :
   - **a.** `To`
   - **b.** `As`
   - **c.** `VarType`

10. Une variable peut être de type :
- **a.** tableau
- **b.** objet
- **c.** état Access

11. Une variable de type objet :
- **a.** contient une référence à l'objet.
- **b.** contient l'objet lui-même.
- **c.** contient la valeur de l'objet.

12. Le type de données numériques le plus précis est :
- **a.** `Single`
- **b.** `Currency`
- **c.** `Double`

13. La conversion des données est possible :
- **a.** oui
- **b.** non
- **c.** uniquement pour les chaînes de caractères.

14. La création de ses propres types de données est possible :
- **a.** oui
- **b.** non
- **c.** seulement pour les tableaux ou collections.

15. Pour connaître le type de données d'une variable, on utilise le mot réservé :
- **a.** `Is`
- **b.** `Cvar`
- **c.** `VarType`

Corrigé p. 175

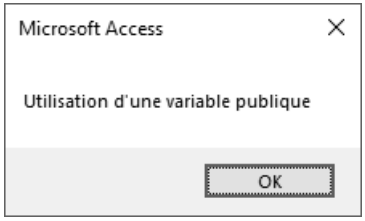
Énoncé 2.1 Déclarer et utiliser une variable

Durée estimative : 20 minutes

2.1.1 : Déclarer et utiliser une variable de niveau projet

Dans le module **Chapitre_02**, déclarez une variable publique nommée **strMessagePublic** de type `String`.

Créez une procédure nommée **Message** qui affiche le message suivant :

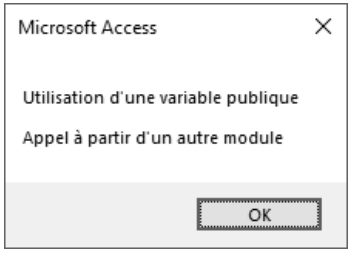


Indice

Pour afficher la boîte de dialogue, utilisez la fonction `MsgBox`.

2.1.2 : Utiliser une variable de niveau projet

Dans le module nommé **Module2**, créez une procédure nommée **AppelExtérieur** qui utilise la variable **strMessagePublic** du module **Chapitre_02** pour afficher le message suivant :



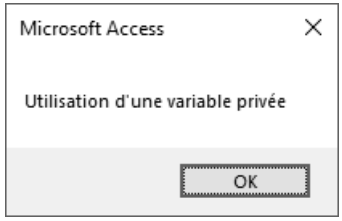
Indices

Pour forcer un saut de ligne : `vbCrLf`.

Utilisez l'opérateur de concaténation & pour composer le message.

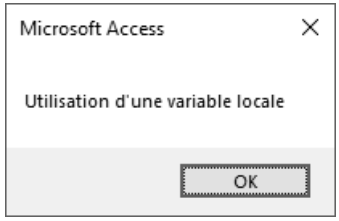
2.1.3 : Déclarer et utiliser une variable de niveau module

Dans le module **Chapitre_02**, déclarez une variable privée nommée **strMessagePrivé** de type `String`.
Créez une procédure nommée **MessageInterne** qui affiche le message suivant :



2.1.4 : Déclarer et utiliser une variable de niveau procédure

Dans le module **Chapitre_02**, créez une procédure nommée **MessageLocal**. Celle-ci utilise une variable locale nommée **strMessageLocal** de type `String`. Résultat :

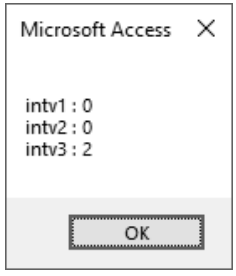


2.1.5 : Déclarer et utiliser plusieurs variables de même type

Dans le module **Chapitre_02**, créez une procédure nommée **VariablesLigne** dont voici le code :

```
Sub VariablesLigne()
 Dim intv1, intv2, intv3 As Integer
 MsgBox ("intv1 : " & VarType(intv1) & vbCrLf & "intv2 : " _
 & VarType(intv2) & vbCrLf & "intv3 : " & VarType(intv3))
End Sub
```

Résultat :



Corrigez la déclaration de sorte que les trois variables soient effectivement de type entier.

Corrigé p. 176

Énoncé 2.2 Déclarer et utiliser une constante

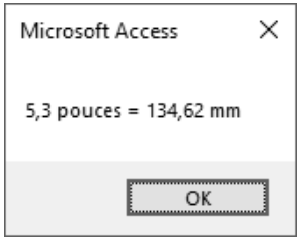
Durée estimative : 10 minutes

2.2.1 : Conversion

Dans le module **Chapitre_02**, déclarez une constante publique nommée **dblPouce** de type `Double`. Affectez-lui la valeur 25,4.

Créez une procédure nommée **ConversionPouceEnmm** qui convertit les pouces en millimètres.

Exemple :



Indice

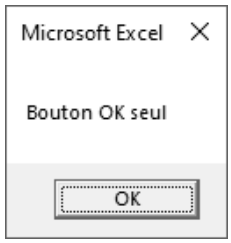
Faites attention au séparateur décimal.

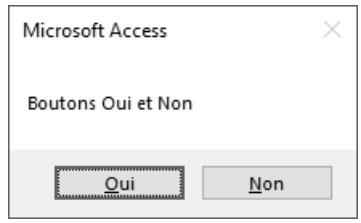
2.2.2 : Utiliser des constantes VBA pour le choix des boutons des boîtes de messages

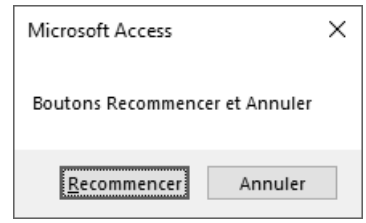
VBA possède de nombreuses constantes internes qui simplifient la programmation. Avec elles, il est possible par exemple de choisir facilement les boutons de la boîte de dialogue standard.

Affichage seulement du bouton **OK** :

```
Sub ConstanteVBA()
 Dim intRésultat As Integer
 intRésultat = MsgBox("Bouton OK seul", vbOKOnly)
End Sub
```







Complétez le code de la procédure **ConstanteVBA** pour obtenir l'affichage des deux autres boîtes de dialogue ci-dessus : la boîte avec les boutons **Oui** et **Non** et la boîte avec les boutons **Recommencer** et **Annuler**.

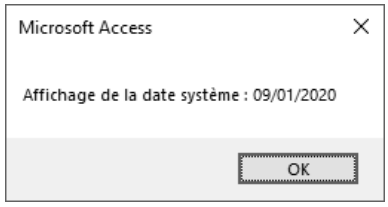
Indice

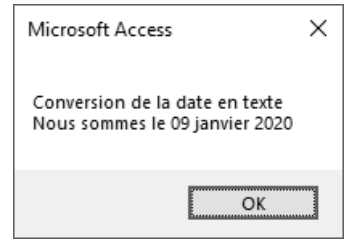
La liste complète des constantes VBA disponibles pour la gestion des boîtes de dialogue MsgBox est accessible par le lien suivant :
https://docs.microsoft.com/fr-fr/office/vba/language/reference/user-interface-help/msgbox-constants

Corrigé p. 177

Énoncé 2.3 Utiliser la date système

Durée estimative : 10 minutes
Créez la procédure **ConversionDate** qui effectue une conversion de la date système en type `String`. Procédez aux deux affichages suivants.
Exemple :





Indices

Utilisez la fonction VBA `Date` *pour avoir la date système.*

Utilisez la fonction VBA `CStr` *pour la conversion.*

Syntaxe : `CStr(expression)`

Utilisez la fonction VBA `Format` *pour la présentation de la date après conversion en texte.*

Syntaxe simplifiée : `Format(expression)`

Corrigé p. 178

Énoncé 2.4 Créer un type de données "PoissonTropical" défini par l'utilisateur

Durée estimative : 5 minutes
Créez dans le module **Chapitre_02** un type de données public nommé **PoissonTropical**. Les membres de ce nouveau type sont : nomPoisson, couleur, poids et lieu d'habitat.

Indice

```
Public . . . PoissonTropical
 NomPoisson As . . .
 Couleur . . .
 Poids . . .
 Lieu . . .
End Type
```

Corrigé p. 179

Énoncé 2.5 Utiliser le type "PoissonTropical"

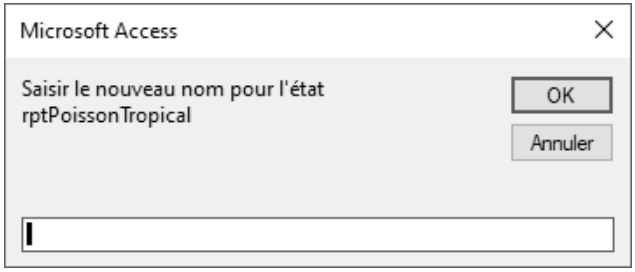
Durée estimative : 15 minutes
Complétez la procédure **NouveauPoissonTropical** ci-après. Elle permet de saisir un nouveau poisson tropical et de l'ajouter comme enregistrement de la table **tblPoissonTropical**.

```
Sub NouveauPoissonTropical()
 Dim rs As DAO.Recordset
 Dim NouveauPoisson As PoissonTropical
 'On Error GoTo Sortie_Sur_Erreur

 ' Ouvrir la table en lecture/écriture
 Set rs = CurrentDb.OpenRecordset("tblPoissonTropical", dbOpenDynaset)
 rs.AddNew

 NouveauPoisson.NomPoisson = ....
 rs("NomPoisson").Value = ...
        .
        .
         .
 rs.Update
 rs.Close
 Set rs = Nothing
Sortie_Sur_Erreur:
End Sub
```

Corrigé p. 179

Énoncé 2.6 Renommer un état Access

Durée estimative : 10 minutes
Créez, dans le module **Chapitre_02**, la procédure **RenommerEtat** qui renomme l'état **lstPoissonTropical** en un nouveau nom saisi par l'utilisateur.



Indice

Utilisez l'instruction `DoCmd.Rename`.

Corrigé p. 180