Introduction à la programmation VB

Conseils utiles :

Les qualités d'un programme

- **Fiabilité** : il doit donner les résultats corrects attendus.
- **Robustesse** : il doit gérer les erreurs de manipulation des utilisateurs.
- **Convivialité** : il doit être agréable à utiliser (souris, icônes, menus…)
- **Efficacité** : il doit donner des réponses rapides et claires.
- **Compacité** : il doit occuper le moins de place possible en mémoire.
- **Lisibilité** : il doit être structuré en modules, commenté, présenté clairement.
- **Portabilité** : il doit être aisément transférable sur une machine d'un autre type.

Méthode de programmation

- Spécification des besoins des futurs utilisateurs.
- Spécifications fonctionnelles : comment satisfaire aux besoins.
- Conception générale : division du logiciel en programmes.
- Conception détaillée : algorithme le plus adapté pour chaque programme.
- Assemblage des différents programmes.
- Codage à l'aide du langage le plus adapté.
- Tests et Validation.

La **conception** est beaucoup plus importante que le **codage** qui peut être sous-traité dans le cas de gros logiciels.

Le langage Visual Basic

Caractéristiques du langage VB

Programmation **événementielle** (sollicitations : souris, clavier, autre événement…)
Réutilisable (modules, classes).

Notion de variable

Elles sont nécessaires pour **stocker** (conserver) une valeur dynamique et réutilisable.

Menu **Outils** - **Options** - onglet **Environnement** - *choisir* : "Requiert la déclaration des variables".

On peut aussi écrire la directive **Option Explicit** au début de la section des déclarations d'un module.

Les types de variables

On recommande fortement de **déclarer** les variables utilisées dans un programme.

- `Integer` : de **-32 768** à **32 767**
- `Long` : de **-2 147 483 648** à **2 147 483 647**
- `Single` : décimaux en simple précision : 39 chiffres significatifs
- `Double` : décimaux en double précision : plus de 300 chiffres significatifs !
- `String` : de **0** à **65 535** octets
- `Variant` : de type nombre , texte ou objet selon l'affectation faite.
- Boolean :de type logique
- …

Syntaxe de déclaration :

```
Dim <NomVariable> As <Type>
```

Pour la lisibilité du code on peut les commenter après une apostrophe (‘)

Exemples :

```
Dim Taux As Single ' Taux de la TVA
Dim Réponse As String ' Mot proposé par le joueur
```

Pour éviter tout problème il est préférable d’initialiser les variables déclarées.

```
Compteur = 0
Taux = 20,6
```

Le langage Basic utilise **7 types** de données dont les plus utilisés sont le type **String** (chaîne de caractères), le type **Integer** (entier relatif) et le type **Single** (décimal).

Portée d'une variable

- Si une variable est déclarée au début de la procédure qui la manipule (**Dim** ou **Private**) elle n’est alors valide que pour cette procédure. L'existence et la valeur de la variable disparaissent avec l'instruction **End Sub**. Toute référence à cette variable en dehors de cette procédure provoquera une erreur de compilation.
- Si une variable est déclarée dans la section des déclarations d’un module elle est valide dans toutes les procédures du module.
- Une variable peut aussi être déclarée **Public** ou **Global** et sera alors valide pour toute l’application.

  Exemple : `Global MotCommun As String`

Les tableaux et les boucles

▶ Structure d'un tableau

On a souvent besoin de travailler sur un ensemble de données.

Un exemple : les températures moyennes des 12 mois de l'année.

On pourrait déclarer **12 variables identiques** :

```
Dim Temp1, Temp2, Temp3, Temp4, … … …, Temp12 as Single
```

On dispose d'une structure de données appelée **Tableau** qui permet de conserver dans une seule "**entité**" plusieurs valeurs de même type.

Le nom du tableau est une variable qu'il est recommandé de préfixer par Tab.

Le nombre de valeurs de types identiques est à déclarer entre parenthèses.

Exemple de déclaration d'un tableau à une dimension :

```
Dim TabTemp(12) As Single
```

Numéro	1	2	3	4	5	…
Température	6	5,5	7	11,5	15	…

L'accés à la case numéro 3 se fait par `TabTemp(3)` qui vaut **7**.

Exemple de déclaration d'un tableau à deux dimensions :

```
Dim TabMajMin(1 to 2,65 to 90) As String
```

Numéro	65	66	67	…	89	90
1	A	B	C	…	Y	Z
2	a	b	c	…	y	z

L'accés à la case qui contient le petit "c"se grâce à la syntaxe suivante : `TabMajMin(2,67)` `qui vaut "c"`.

▶ Boucles en nombre défini

Cette boucle est utilisée lorqu'on connaît à l'avance le nombre de fois qu'elle sera parcourue.

Syntaxe :

```
For Compteur = Début To Fin [Step Incrément]
Instructions
[ ... Exit For]
[Instructions]
Next [Compteur]
```

Le test est effectué **au début** de la boucle.

La variable numérique Compteur est **incrémentée** à chaque fin de boucle du nombre indiqué par l'incrément. Si l'incrément n'est pas spécifié il est fixé à **1**.

Si la valeur de **Fin** est inférieure à la valeur de **Début** l'incrément est négatif.

La valeur de **Compteur** peut être utilisée (par exemple pour numéroter le passage dans la boucle) mais ne doit pas être modifiée dans le corps de la boucle.

Exemple :

```
For i = 1 To 50
TabInitial(i) = 0 ' Initialisation de chaque case à 0
Next i
```

- Remplissage d'un tableau

Pour remplir un tableau on le balaye avec une boucle **For … To … Next** (car le nombre de cases est connu à l'avance).

Exemple 1 :

```
Dim TabTemp(12) As Single
Dim Compteur As Integer
For Compteur = 1 To 12
     TabTemp(Compteur)=InputBox("Température N° " & Compteur)
Next Compteur
```

Exemple 2 :

```
Dim TabTirageLoto(6) As Integer
Dim Compteur As Integer
For Compteur = 1 To 6
     TabTirageLoto (Compteur)=Rnd * 48 + 1
Next Compteur
```

Les boucles :

Si le programme doit exécuter un bloc d'instructions en nombre prédéfini on utilise la boucle `For … To … Next.`

Exemple :

```
For i = 1 To 49
                TabLoto(i) = i 'chaque case contient son numéro
Next i
```

Si le nombre de passages dans la boucle est inconnu au départ, mais dépend d'une **condition** dont la réalisation est imprévisible cette structure n'est pas adaptée.

Exemple 1 :

Demander le mot de passe **tant que** la réponse n'est pas le bon mot de passe.

Demander le mot de passe **jusqu'à ce que** la réponse soit le bon mot de passe.

Exemple 2 :

Demander la saisie d'une note **tant que** la réponse n'est pas un nombre entre 0 et 20.

Demander la saisie d'une note **jusqu'à ce que** la réponse soit un nombre entre 0 et 20.

Boucle tant que

Syntaxe première version :

```
Do While Condition
Instructions
[... Exit Do]
[Instructions]
Loop
```

La condition est ici testée **au début** c'est à dire à l'entrée de la boucle.

Avec `While` (tant que) la boucle est répétée **tant que** la condition est **vraie**.

Si la condition n'est pas vraie au départ les instructions de la boucle ne sont pas exécutées.

Exemple :

```
Do While MotProposé <> MotDePasse
MotProposé = InputBox("Donnez votre mot de passe")
Loop
```

Cela présuppose MotProposé initialisé par une valeur autre que MotDePasse

(par exemple la valeur par défaut "").

Syntaxe deuxième version :

```
Do
Instructions
[... Exit Do]
[Instructions]
Loop While Condition
```

La condition est alors testée **à la fin** de la boucle.

Avec `While` (tant que) la boucle est répétée **tant que** la condition est **vraie**.

Les instructions de la boucle sont donc exécutées au moins une fois.

Exemple :

```
Do
MotProposé = InputBox("Donnez votre mot de passe")
Loop While MotProposé <> MotDePasse
```

Cet exemple ne présuppose aucune initialisation de MotProposé.

Les boucles Tant que , jusqu'à

Syntaxe première version :

```
Do while Condition
Instructions
[... Exit Do]
[Instructions]
Loop
```

La condition est ici testée **au début** c'est à dire à l'entrée de la boucle.

Avec `while` (tant que) la boucle est répétée **tant que** la condition n'est pas **vérifiée**.

Si la condition est vraie dès le départ les instructions de la boucle ne sont pas exécutées.

Exemple :

```
Do while MotProposé <> MotDePasse
MotProposé = InputBox("Donnez votre mot de passe")
Loop
```

Cela présuppose MotProposé initialisé par une valeur autre que MotDePasse

(par exemple la valeur par défaut : "").

Syntaxe deuxième version :

```
Do
Instructions
[... Exit Do]
[Instructions]
Loop Until Condition
```

La condition est alors testée **à la fin** de la boucle.

Les instructions de la boucle sont donc exécutées au moins une fois.

Avec Until (jusqu'à) la boucle est répétée **jusqu'à ce que** la condition soit **vraie**.

Exemple :

```
Do
MotProposé = InputBox("Donnez votre mot de passe")
Loop Until MotProposé = MotDePasse
```

Cet exemple ne présuppose aucune initialisation de MotProposé.

Boucle For … Each … Next

C'est une extension de la boucle For ... To … Next. Elle est utilisée pour parcourir les collections(ensembles).

Syntaxe :

```
For Each Elément In Ensemble
Instructions
[ ... Exit For]
[Instructions]
Next [Elément]
```

Ensemble est le plus souvent un tableau.

Exemple :

```
Dim TabHasard(100) As Integer
Dim Cellule As Integer
Dim Réponse As String
Randomize
For Each Cellule In TabHasard
                Cellule = Rnd * 100 + 1
Next
For Each Cellule In TabHasard
Réponse = Réponse & Cellule & "   "
Next
MsgBox (Réponse)
```

Traitement des chaînes de caractères

Nécessité de ces traitements

Les données manipulées par un programme sont essentiellement de type **numérique** ou **chaîne de caractères**.

Si les types numériques sont très utilisés par les programmes scientifiques, le type chaîne est incontournable pour des étudiants en Lettres et Sciences Humaines.

Une variable chaîne de caractères se déclare de type String.

Exemple 1 :

```
Dim MotProposé As String
```

La variable contient alors une chaîne de longueur **variable** selon l'affectation qui suivra.

Exemple 2 :

```
Dim Lettre As String * 1
```

La variable contient alors une chaîne de longueur 1 c'est à dire **un seul caractère**.

Exemple 3 :

```
Dim Adresse As String * 30
```

La variable contient alors une chaîne de **longueur 30**. Si l'on n'affecte que 18 caractères dans une telle chaîne, le reste est rempli d'espaces. Si l'on affecte plus de 30 caractères le surplus est tronqué.

Comparaison des chaînes de caractères

Longueur d'une chaîne :

La longueur d'une chaîne est donnée par la fonction **Len**.

Exemple :

```
Phrase = "Visual Basic"
```

La fonction **`Len(Phrase)`** retournera la valeur **12**.

Il est évident que si deux chaînes de caractères n'ont pas la même longueur elles sont différentes. Par contre deux chaînes de même longueur ne sont pas forcément identiques.

Comparaison binaire :

On compare deux chaînes par la fonction **StrComp**.

Elle renvoie la valeur numérique **0** si les deux chaînes sont rigoureusement identiques et la valeur numérique **1** si les chaînes diffèrent même par un seul octet.

Exemple :

```
Phrase1 = "Visual Basic."

Phrase2 = "Visual basic."

Phrase3 = " Visual Basic."
```

La fonction **StrComp(Phrase1,Phrase2)** retournera la valeur …

La fonction **StrComp(Phrase1,Phrase3)** retournera la valeur …

Recherche d'une chaîne de caractères

- La fonction InStr permet de rechercher si une chaîne de caractères **existe** à l'intérieur d'une autre chaîne de caractères. Si c'est le cas, elle retourne la **position** de la première occurrence de la chaîne recherchée.

  Syntaxe :

  ```
  InStr(Chaîne1, Chaîne2)
  ```

  Chaîne1 est la chaîne de caractères à traiter (sur laquelle porte la recherche).

  Chaîne2 est la chaîne de caractères recherchée dans **Chaîne1**.

  Exemple :

  ```
  Chaîne1 = "Visual Basic et ses fonctions"

  Chaîne2 = "Basic"

  Chaîne3 = "Basics"
  ```

  La fonction **InStr(Chaîne1,Chaîne2)** retournera la valeur **8**

  La fonction **InStr(Chaîne1,Chaîne3)** retournera la valeur **0**.

- La fonction Ucase met tout le texte en majuscules et permet de rechercher indépendamment une lettre minuscule ou majuscule.
- La fonction Lcase met tout le texte en minuscules et permet…

Extraction d'une chaîne de caractères

- La fonction Right donne la partie **droite** d'une chaîne de caractères. Le nombre de caractères de cette partie doit être précisé.

Exemple :

```
Chaîne = "Visual Basic et ses fonctions"
```

La fonction **`Right(Chaîne,5)`** retournera la valeur "`tions`"

- La fonction Left donne la partie **gauche** d'une chaîne de caractères. Le nombre de caractères de cette partie doit être précisé.

  Exemple :

  ```
  Chaîne = "Visual Basic et ses fonctions"
  ```

  La fonction **`Left(Chaîne,5)`** retournera la valeur "`Visua`"

- La fonction Mid extrait une **partie** d'une chaîne de caractères.

  Le nombre de caractères de cette partie doit être précisé ainsi que la position du premier caractère extrait.

  Syntaxe :

  ```
  Mid(Texte, Position, Nombre)
  ```

  `Texte` est la chaîne de caractère à traiter.

  `Position` est la position du caractère à partir duquel il faut extraire une sous-chaîne de caractères.

  `Nombre` est le nombre de caractères à extraire.

  Exemple :

  ```
  Chaîne = "Visual Basic et ses fonctions"
  ```

  La fonction **`Mid(Chaîne,8,5)`** retournera la valeur "`Basic`"

  La fonction **`Mid(Chaîne,21)`** retournera la valeur "`fonctions`"

  Applications

  La plupart des problèmes sur les chaînes de caractères se traitent à l'aide des fonctions ci-dessus. Leur maniement est assez délicat et les résultats quelquefois difficiles à prévoir. Seule la pratique de leur programmation vous permettra d'en vérifier l'efficacité.

Exemple1 :

Ce programme recherche un mot dans une chaîne de caractères :

```
Dim Chaine, Mot As String
Dim Position As Integer
Chaine = "Visual Basic et ses fonctions."
Mot = InputBox("Taper le mot à rechercher")
Position = InStr(Chaine, Mot)
If Position = 0 Then
     MsgBox "Le mot '" & Mot & "'" & " n'a pas été introuvé !")
Else
 MsgBox("Le mot '" & Mot & "'"&" a été trouvé à la position "_
          & Position)
End If
```

Exemple2 :

Ce programme compte le nombre d'espaces dans une phrase.

```
     Dim Phrase, Caractère As String
     Dim Compteur, Longueur, i As Integer
     Phrase = InputBox("Tapez votre phrase")
     Longueur = Len (Phrase)
     For i = 1 To Longueur
               Caractère = Mid(Phrase, i, 1)
               If Caractère = " " Then Compteur = Compteur + 1
      Next i
     MsgBox("Cette phrase contient " & Compteur & " espaces.")
```

Fonctions de date et d'heure

- La fonction **Date** donne la date système.

  `Date` retourne la date du jour courant.

- La fonction **Time** donne l'heure système.

`Time` retourne l'heure courante.

- La fonction **Day()** donne le numéro du jour dans le mois.

  `Day(Date)` retourne le numéro du jour du mois courant (compris entre 1 et 31)a valeur **8**

- La fonction **Month()** donne le numéro du mois dans l'année.

  `Month(Date)` retournera le numéro du mois courant(compris entre 1 et 12). Ex : **11.**

- La fonction **Year()** donne le numéro de l'année.

  `Year(Date)` retournera le numéro de l'année courante. Ex : **2000.**

- La fonction **WeekDay()** donne le numéro du jour dans la semaine sachant que le dimanche porte le numéro 1.

  `WeekDay(Date)` retournera le numéro d'aujourd'hui (compris entre 1 et 7) .

- La fonction **FormatDateTime(date [, constante vb])** donne Retourne une expression formatée sous forme de date ou d'heure. Les crochets s.

  Où constante vb peut prendre les valeurs suivantes **: [, vGeneralDate ou vbLongDate, vbShortDate, vbShortTime, …]**

Autres fonctions

1) Get Déclare le nom, les arguments et le code formant le corps d'une procédure **Property**, qui lit la valeur d'une propriété.

Syntaxe:

[**Public** | **Private** | **Friend**] [**Static**] **Property Get** *NomPropriété* [**(***arglist***)**] [**As** *type*]

[*instructions*]

[**Exit Property**]
[*instructions*]

End Property

2) **Let** : Déclare le nom, les arguments et le code formant le corps d'une procédure **Property Let**, qui attribue une valeur à une propriété.

Syntaxe

[**Public** | **Private** | **Friend**] [**Static**] **Property Let** *name* **(**[*arglist***,**] *value***)**
[*instructions*]
[**Exit Property**]
[*instructions*]

End Property

Exemple

```
Private dblBalance As Double

Public Property Get Balance() As Double
    Balance = dblBalance
End Property

Property Let Balance(dblNewBalance As Double)
    dblBalance = dblNewBalance
End Property
```

4) CallByName : cette fonction permet d'obtenir ou de définir une propriété ou d'invoquer une méthode pendant l'exécution.

Syntaxe : **CallByName**(**Objet, NomProcédure,CallType**)

Où :

- Objet(objet variant) est le nom de l'objet sur lequel on souhaite exécuter la fonction **CallByName**.
- NomProcédure (variant) désigne le nom de la propriété ou de la méthode de l'objet .
- CallType est une constante qui représente le type de procédure appelée(vbGet, VbLet,vbMethod).

Exemple :

- *Obtenir le contenu de la propriété NomGroupe de l'objet objGroupe :*

  Resulat = CallByName(objgroupe, "NomGroupe", VbGet)

- *Définir la propriété NomGroupe de l'objet objGroupe :*

  CallByName(objgroupe, ObjField.Name, VbLet)

- Exécuter la méthode "InitialisationObj" *de l'objet objGroupe :*

  CallByName(objgroupe, "InitialisationObj", Vbmethod)

Procédures et fonctions

Visual Basic permet de définir et d'utiliser trois types de sous-routines :

Les procédures proprement dites :

Une procédure est un ensemble d'instructions qui traite une tâche donnée.

Elle débute par le mot réservé **Sub** et se termine par **End Sub**.

Exemple de procédure événementielle :

```
Private Sub cmdQuitter_Click
     End
End Sub
```

Si un bloc d'instructions doit être utilisé à plusieurs endroits (par exemple dans plusieurs procédures événementielles) il est préférable d'en faire une **procédure publique** qui sera pourra être appelée dans n'importe quel autre sous-programme du module.

Exemple de procédure publique :

```
Public Sub SaisieNote()
Do
Note = InputBox("Tapez une note")
Loop Until Note>=0 And Note<=20
End Sub
```

Pour utiliser cette procédure il suffira de l'appeler par son nom : `SaisieNote`

Les fonctions :

Elles sont comparables aux procédures quant à leur mode d'écriture. La différence ce qu'une fonction débute par le mot réservé **Function** et se termine par **End Function**. De plus, elle a un type et peut retourner **une valeur** contenue dans le nom même de la fonction. Cette valeur pourra être utilisée par la suite dans une autre expression.

Exemple de fonction :

```
Public Function Carré(x) As Single
     Carré = x * x
End Function
```

Par exemple `Carré(7)` retournera la valeur **49**.

Les procédures Property :

Elles servent à définir les propriétés d'un objet par exemple dans un module, dans une classe ou dans une feuille. Elles sont délimitée par les mots clés **Property** et **End Property.**

Exemple de propriété :

```
Public Property Let Nom(ByVal parametre As String)
    mvarNom = parametre
End Property
```

Remarques :

ByVal	Indique que l'argument est passé par valeur.
ByRef(valeur par défaut).	Indique que l'argument est passé par référence

Interface de développement de VISUAL BASIC

L'éditeur de code de Visual Basic

Composé de 3 parties :

1. La partie supérieure est la zone des **déclarations** (séparée du reste par un trait horizontal). On y place les options et les déclarations de variables publiques.

   Exemple :

```
Option Explicit
Public NomJoueur As String
Private PrenomJour as String
```

2. La partie suivante est la zone des **procédures publiques** (chaque procédure est séparée des autres par un trait horizontal). On y place les procédures publiques utilisables par toutes les autresprocédures du module.

   Exemple :

```
Public Function Carré(x) As Single
Carré = x * x
End Function
```

3. Enfin la partie suivante est la zone des **procédures événementielles** (chaque procédure est séparée des autres par un trait horizontal).

   Exemple :

```
Private Sub cmdCalculer_Click()
End Sub
```

Algorithmique

Quand on a un problème à résoudre par programmation on doit tout d'abord trouver une **stratégie** pour y parvenir.

Celle-ci doit bien sûr être "**programmable**" dans un langage de programmation. Il faut donc bien connaître les caractéristiques et les possibilités de ce langage.

Le plus souvent on écrit un **algorithme** en français (c'est la stratégie adoptée) que l'on pourra ensuite **coder** dans le langage de programmation choisi.

***Exemple1* :**

Déterminer la carte la plus forte sur un ensemble de 6 cartes posées à l'endroit sur la table.

La résolution de ce problème par un être humain ou par un programme informatique est complètement différente.

Algorithme en français

Prendre la 1ere carte. Noter sa hauteur dans une variable.

De la 2eme à la dernière :

    Prendre une carte.

    Si sa hauteur est supérieure à celle notée dans la variable elle devient la plus forte

Recommencer

***Exemple2* :**

Trier en ordre décroissant les nombre contenus dans un tableau de 100 entiers.

Différentes stratégies sont à notre disposition. La plus classique est celle du **tri à bulles**.

Il s'agit de comparer chaque nombre à son suivant et de mettre en premier le plus grand des deux. On recommence ainsi jusqu'à ce que plus aucune permutation ne soit effectuée. Alors le tableau est trié.

Algorithme en français

Début

Faire

Permuté ← 0

    Pour i = 1 à 99

        Si Case i < Case i+1 alors

            Auxiliaire ← Case i

            Case i ← Case i+1

Case i+1 ← Auxiliaire

Permuté ← 1

Fin de Si

Nouveau i

Recommencer tant que Permuté = 1

Fin

Exemple d'utilisation des méthodes Execute, Requery et Clear

L'exemple ci-dessous décrit la méthode **Execute** exécutée à partir d'un objet **Command** et d'un objet **Connection**. Il utilise également la méthode **Requery** pour récupérer les données en cours dans un jeu d'enregistrements, puis la méthode **Clear** pour supprimer le contenu de la collection **Errors**. Les procédures ExecuteCommand et PrintOutput sont nécessaires à l'exécution de cette procédure.

```
Public Sub ExecuteX()

   Dim strSQLChange As String
   Dim strSQLRestore As String
   Dim strCnn As String
   Dim cnn1 As ADODB.Connection
   Dim cmdChange As ADODB.Command
   Dim rstTitles As ADODB.Recordset
   Dim errLoop As ADODB.Error

   ' Définit deux instructions SQL à exécuter en tant que texte de
commande.
   strSQLChange = "UPDATE Titles SET Type = " & _
      "'self_help' WHERE Type = 'psychology'"
   strSQLRestore = "UPDATE Titles SET Type = " & _
      "'psychology' WHERE Type = 'self_help'"
   ' Ouvre une connexion.
   strCnn = "Provider=sqloledb;" & _
      "Data Source=srv;Initial Catalog=pubs;User Id=sa;Password=; "
   Set cnn1 = New ADODB.Connection
   cnn1.Open strCnn

   ' Crée un objet de commande.
   Set cmdChange = New ADODB.Command
   Set cmdChange.ActiveConnection = cnn1
   cmdChange.CommandText = strSQLChange

   ' Ouvre la table des titres.
   Set rstTitles = New ADODB.Recordset
   rstTitles.Open "titles", cnn1, , , adCmdTable

   ' Imprime l'état des données initiales.
   Debug.Print _
      "Data in Titles table before executing the query"
   PrintOutput rstTitles
   ' Supprime les autres erreurs de la collection Errors.
   cnn1.Errors.Clear
   ' Appelle la sous-routine ExecuteCommand pour
   ' exécuter la commande cmdChange.
   ExecuteCommand cmdChange, rstTitles
```

```
   ' Imprime l'état des nouvelles données.
   Debug.Print _
      "Data in Titles table after executing the query"
   PrintOutput rstTitles
   ' Utilise la méthode d'exécution de l'objet Connection pour
   ' exécuter des instructions SQL de restauration des données.
' Intercepte les erreurs, en vérifiant la collection Errors si nécessaire.
   On Error GoTo Err_Execute
   cnn1.Execute strSQLRestore, , adExecuteNoRecords
   On Error GoTo 0

   ' Récupère les données en cours en interrogeant de nouveau
   ' le jeu d'enregistrements.
   rstTitles.Requery

   ' Imprime l'état des données restaurées.
   Debug.Print "Data after executing the query " & _
      "to restore the original information"
   PrintOutput rstTitles
   rstTitles.Close
   cnn1.Close

   Exit Sub
   Err_Execute:
   ' Avertit l'utilisateur des éventuelles erreurs pouvant résulter de
   ' l'exécution de la requête.
   If Errors.Count > 0 Then
      For Each errLoop In Errors
         MsgBox "Error number: " & errLoop.Number & vbCr & _
         errLoop.Description
      Next errLoop
   End If
   Resume Next
End Sub

Public Sub ExecuteCommand(cmdTemp As ADODB.Command, _
   rstTemp As ADODB.Recordset)
   Dim errLoop As Error
   ' Exécute l'objet Command spécifié. Intercepte les
   ' erreurs, en vérifiant la collection Errors si nécessaire.
   On Error GoTo Err_Execute
   cmdTemp.Execute
   On Error GoTo 0

   ' Récupère les données en cours en interrogeant de nouveau
   ' le jeu d'enregistrements.
   rstTemp.Requery

   Exit Sub
   Err_Execute:
   ' Avertit l'utilisateur des éventuelles erreurs pouvant résulter de
   ' l'exécution de la requête.
   If Errors.Count > 0 Then
      For Each errLoop In Errors
         MsgBox "Error number: " & errLoop.Number & vbCr & _
         errLoop.Description
      Next errLoop
   End If

   Resume Next
End Sub
```

```
Public Sub PrintOutput(rstTemp As ADODB.Recordset)
   ' Répertorie le Recordset.
   Do While Not rstTemp.EOF
      Debug.Print "  " & rstTemp!Title & _
         ", " & rstTemp!Type
      rstTemp.MoveNext
   Loop
End Sub
```

Version VBScript

Voici le même exemple, écrit en VBScript pour pouvoir être utilisé dans une page ASP. Pour visualiser cet exemple fonctionnel, vous devez créer un DSN appelé AdvWorks à l'aide de la source de données AdvWorks.mdb installée avec IIS et située dans C:\InetPub\ASPSamp\AdvWorks. Il s'agit d'un fichier de base de données Microsoft Access. Utilisez **Find** pour retrouver le fichier Adovbs.inc et placez-le dans le répertoire que vous souhaitez utiliser. Coupez et collez le code ci-dessous dans le Bloc-notes ou un autre éditeur de texte, puis sauvegardez-le sous le nom Execute.asp. Vous pouvez visualiser le résultat sous n'importe quel navigateur client.

```
<!-- #Include file="ADOVBS.INC" -->
<HTML><HEAD>
<TITLE>ADO Execute Method</TITLE></HEAD>

<BODY>
<FONT FACE="MS SANS SERIF" SIZE=2>
<Center><H3>ADO Execute Method</H3><H4>Recordset Retrieved Using Connection
Object</H4>
<TABLE WIDTH=600 BORDER=0>
<TD VALIGN=TOP ALIGN=LEFT COLSPAN=3><FONT SIZE=2>

<!--- Recordsets retrieved using Execute method of Connection and Command
Objects-->
<%
Set OBJdbConnection = Server.CreateObject("ADODB.Connection")
OBJdbConnection.Open "AdvWorks"
SQLQuery = "SELECT * FROM Customers"
'Premier Recordset RSCustomerList
Set RSCustomerList = OBJdbConnection.Execute(SQLQuery)

Set OBJdbCommand = Server.CreateObject("ADODB.Command")
OBJdbCommand.ActiveConnection = OBJdbConnection
SQLQuery2 = "SELECT * From Products"
OBJdbCommand.CommandText = SQLQuery2
Set RsProductList = OBJdbCommand.Execute

%>
<TABLE COLSPAN=8 CELLPADDING=5 BORDER=0>
<!-- BEGIN column header row for Customer Table-->
<TR><TD ALIGN=CENTER BGCOLOR="#008080">
<FONT STYLE="ARIAL NARROW" COLOR="#ffffff" SIZE=1>Company Name</FONT>
</TD>
<TD ALIGN=CENTER BGCOLOR="#008080">
<FONT STYLE="ARIAL NARROW" COLOR="#ffffff" SIZE=1>Contact Name</FONT>
</TD>
<TD ALIGN=CENTER WIDTH=150 BGCOLOR="#008080">
<FONT STYLE="ARIAL NARROW" COLOR="#ffffff" SIZE=1>E-mail address</FONT>
</TD>
<TD ALIGN=CENTER BGCOLOR="#008080">
<FONT STYLE="ARIAL NARROW" COLOR="#ffffff" SIZE=1>City</FONT>
</TD>
```

```
<TD ALIGN=CENTER BGCOLOR="#008080">
<FONT STYLE="ARIAL NARROW" COLOR="#ffffff" SIZE=1>State/Province</FONT>
</TD></TR>

<!--Display ADO Data from Customer Table-->
<% Do While Not RScustomerList.EOF %>
   <TR>
   <TD BGCOLOR="f7efde" ALIGN=CENTER>
   <FONT STYLE="ARIAL NARROW" SIZE=1>
   <%= RSCustomerList("CompanyName")%>
   </FONT></TD>
   <TD BGCOLOR="f7efde" ALIGN=CENTER>
   <FONT STYLE="ARIAL NARROW" SIZE=1>
   <%= RScustomerList("ContactLastName") & ", " %>
   <%= RScustomerList("ContactFirstName") %>
   </FONT></TD>
   <TD BGCOLOR="f7efde" ALIGN=CENTER>
   <FONT STYLE="ARIAL NARROW" SIZE=1>

   <%= RScustomerList("ContactLastName")%>
   </FONT></TD>
   <TD BGCOLOR="f7efde" ALIGN=CENTER>
   <FONT STYLE="ARIAL NARROW" SIZE=1>
   <%= RScustomerList("City")%>
   </FONT></TD>
   <TD BGCOLOR="f7efde" ALIGN=CENTER>
   <FONT STYLE="ARIAL NARROW" SIZE=1>
   <%= RScustomerList("StateOrProvince")%>
   </FONT></TD>
   </TR>

<!-Next Row = Record Loop and add to html table-->
<%
RScustomerList.MoveNext
Loop
RScustomerList.Close

%>

</TABLE><HR>
<H4>Recordset Retrieved Using Command Object</H4>
<TABLE COLSPAN=8 CELLPADDING=5 BORDER=0>
<!-- BEGIN column header row for Product List Table-->
<TR><TD ALIGN=CENTER BGCOLOR="#800000">
<FONT STYLE="ARIAL NARROW" COLOR="#ffffff" SIZE=1>Product Type</FONT>
</TD>
<TD ALIGN=CENTER BGCOLOR="#800000">
<FONT STYLE="ARIAL NARROW" COLOR="#ffffff" SIZE=1>Product Name</FONT>
</TD>
<TD ALIGN=CENTER WIDTH=350 BGCOLOR="#800000">
<FONT STYLE="ARIAL NARROW" COLOR="#ffffff" SIZE=1>Product
Description</FONT>
</TD>
<TD ALIGN=CENTER BGCOLOR="#800000">
<FONT STYLE="ARIAL NARROW" COLOR="#ffffff" SIZE=1>Unit Price</FONT>
</TD></TR>

<!-- Display ADO Data Product List-->
<% Do While Not RsProductList.EOF %>
   <TR>
   <TD BGCOLOR="f7efde" ALIGN=CENTER>
```

```
   <FONT STYLE="ARIAL NARROW" SIZE=1>
   <%= RsProductList("ProductType")%>
   </FONT></TD>
   <TD BGCOLOR="f7efde" ALIGN=CENTER>
   <FONT STYLE="ARIAL NARROW" SIZE=1>
   <%= RsProductList("ProductName")%>
   </FONT></TD>
   <TD BGCOLOR="f7efde" ALIGN=CENTER>
   <FONT STYLE="ARIAL NARROW" SIZE=1>
   <%= RsProductList("ProductDescription")%>
   </FONT></TD>
   <TD BGCOLOR="f7efde" ALIGN=CENTER>
   <FONT STYLE="ARIAL NARROW" SIZE=1>
   <%= RsProductList("UnitPrice")%>
   </FONT></TD>

<!--  Prochaine ligne = Enregistrement -->
<%
RsProductList.MoveNext
Loop
'Supprime des objets de la mémoire pour libérer des ressources
RsProductList.Close
OBJdbConnection.Close
Set ObJdbCommand = Nothing
Set RsProductList = Nothing
Set OBJdbConnection = Nothing
%>
</TABLE></FONT></Center></BODY></HTML>
```