Premiers pas en programmation

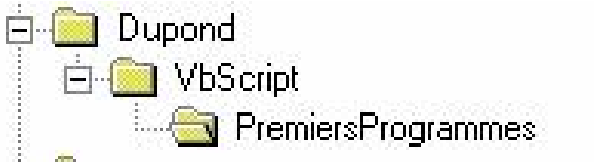
Langage utilisé : vbscript

TABLE DES MATIERES

PREPARATION DE SON ENVIRONNEMENT DE TRAVAIL

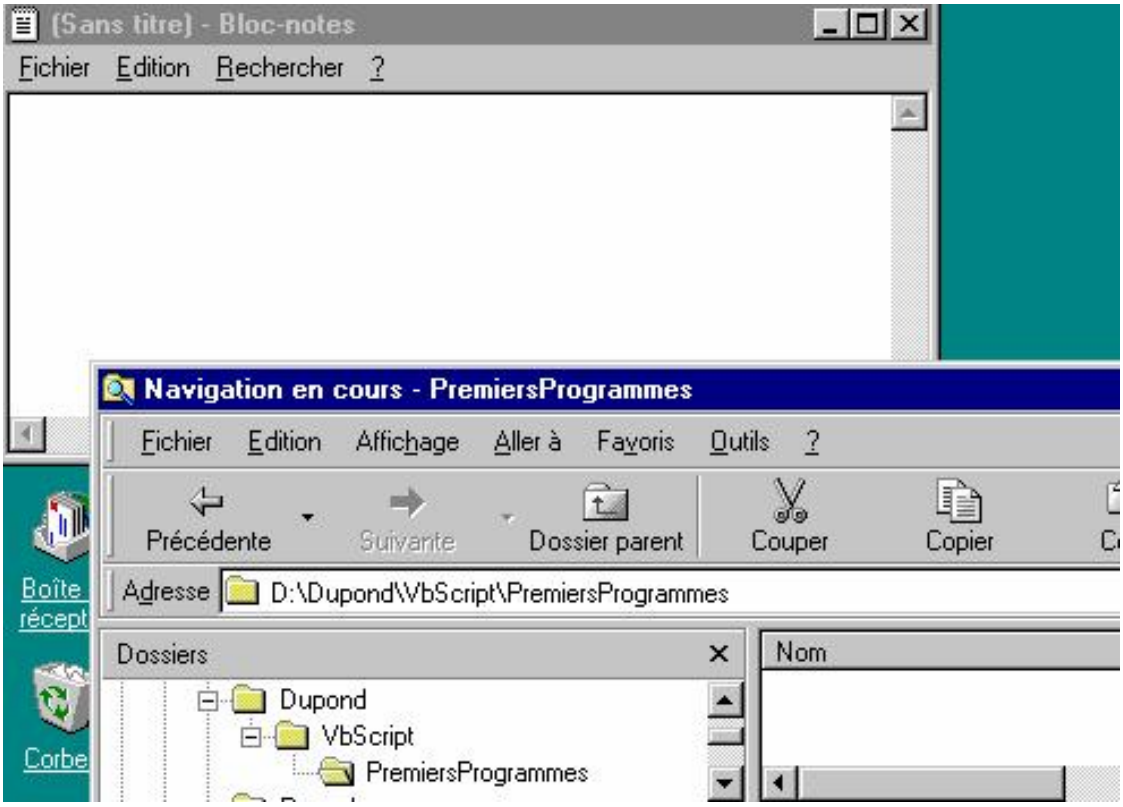
Dans le répertoire de travail (dont le nom est identique à votre nom de famille), créer un sous-répertoire nommé « VbScript »

Dans le répertoire « VbScript », créer un sous- répertoire « PremiersProgrammes ».



Ouvrir le notepad.

Sur votre écran, ne laisser ouvert que l'explorateur de fichier et le notepad, de façon à correspondre avec l'image suivante



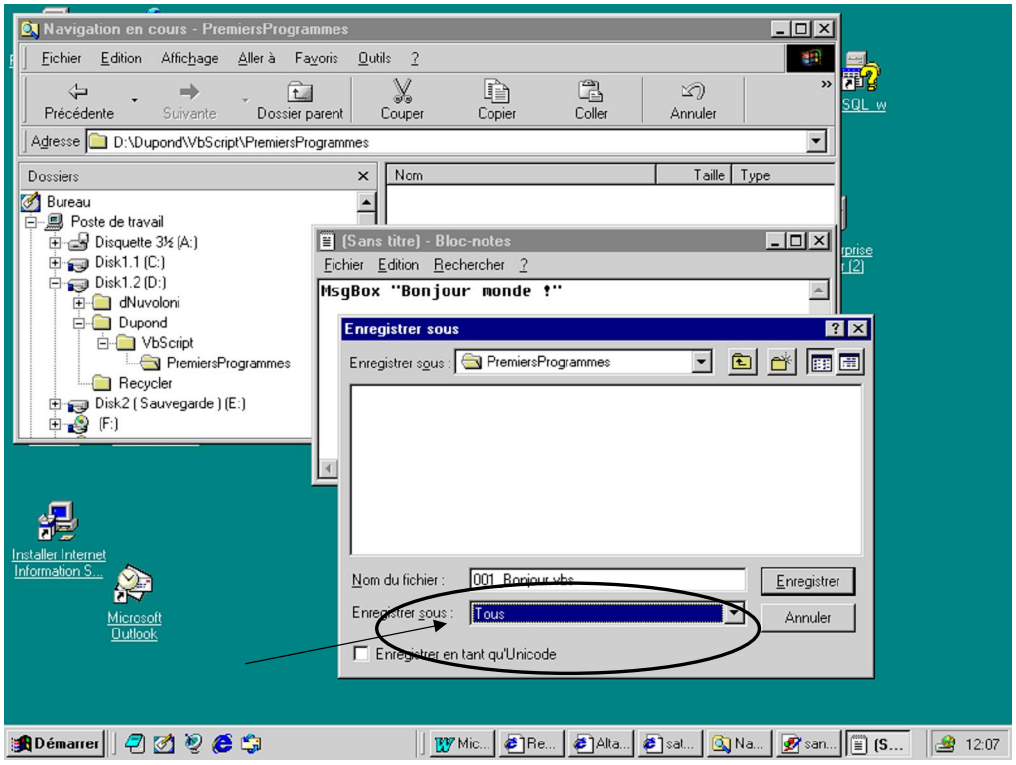
é LE CRI DU NOUVEAU NE

Notre premier programme consistera à annoncer notre venue dans le monde merveilleux de la programmation.

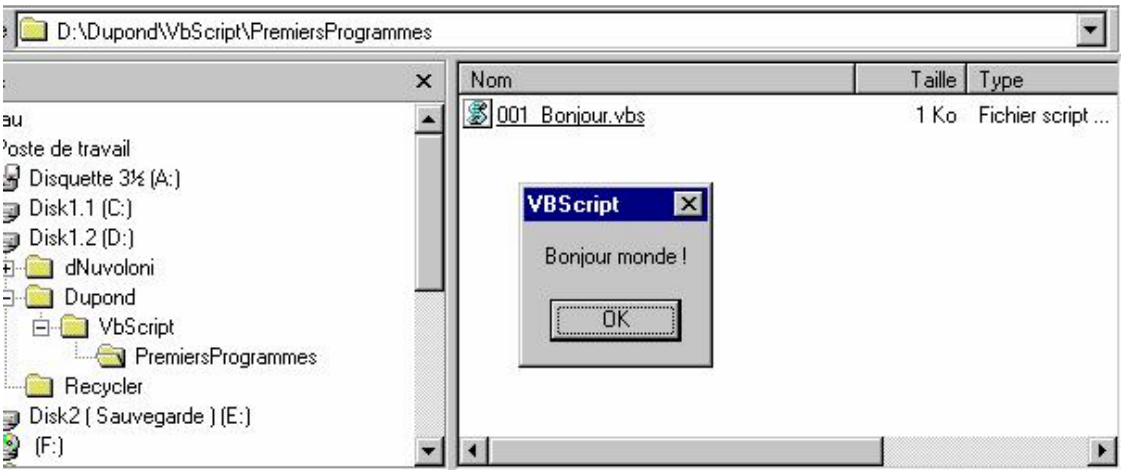
Dans le notepad, taper

```
MsgBox "Bonjour monde !"
```

Enregistrer le fichier sous le nom « 001_Bonjour.vbs »



Dans le sous répertoire « [votrenom]\VbScript\PremiersProgrammes » vous devez voir maintenant le fichier « 001_Bonjour.vbs ».
Cliquer dessus pour voir le résultat.



DISCUSSION :
Le système a détecté qu'il s'agissait d'un fichier contenant du code vbscript grâce à l'extension (.vbs).
Il a appelé un programme (WSCRIPT.EXE, mais peu importe son nom) dont la tâche est d'interpréter le code vbscript et de respecter ses instructions (comme Msgbox).

Entre le click sur le fichier « 001_Bonjour.vbs » et le résultat final, de nombreuses couches logicielles sont utilisées ; mais aujourd'hui il est inutile d'en connaître le détail.
Peu de personnes peuvent d'ailleurs se targuer de pouvoir les lister et quelles sont leurs significations exactes.

Qu'importe les couches basses, seul le résultat compte !

Demain, ces couches peuvent changer, évoluer, offrir de nouvelles fonctionnalités ; l'avantage de manipuler un langage de « haut niveau » est de pouvoir en faire abstraction et donc de pérenniser son savoir faire.

LES PARAMETRES

Dans le titre de la fenêtre ou s'affichait « Bonjour Monde », on pouvait lire « VBScript ».
Il est possible d'y placer des caractères différents.
Voici la syntaxe pour modifier le titre de la fenêtre.

Taper la ligne de code suivante dans un nouveau fichier nommé « 002_Bonjour.vbs » et lancer le programme en cliquant dessus via l'explorateur de fichier.
(Attention à ne pas oublier les deux virgules entre les **chaînes de caractères**)


```
MsgBox "Bonjour monde !",,"Ceci est le titre"
```

Fichier 002_Bonjour.vbs

DISCUSSION :
Nous avons demandé à l'instruction Msgbox de placer un titre.
Pour cela, nous avons placé comme troisième **paramètre** une **chaîne de caractères.**
L'instruction Msgbox admet 3 paramètres.
le deuxième paramètre n'a pas été précisé, voilà pourquoi on peut remarquer deux virgules accolées.
Si nous ne les avions pas mis, VbScript aurait interprété « ceci est un titre » comme deuxième paramètre, et aurait eu des difficultés à en interpréter le sens.

Lire attentivement la rubrique d'aide « MsgBox, fonction ».

Créer un sous répertoire « [votrenom]\VbScript\MesTests ».
Créer au moins 4 petits programmes concernant l'instruction Msgbox , relatifs aux différentes explications et exemples du fichiers d'aide, qui seront placés dans le sous-repertoire « MesTests ».

LES COMMENTAIRES

A une certaine époque (pas si éloignée é), les programmeurs avaient la responsabilité dûne application de A à Z.
Ils développaient dans leur coin, et devaient **maintenir** (c.a.d faire évoluer en fonction des nouveaux besoins de lûutilisateur) leurs différents programmes.
Lorsquûils nûétaient plus là, le programme ne pouvait tout simplement plus évoluer, car eux seuls étaient capables (dans le meilleur des cas !) de le modifier.
Lorsque les programmes contiennent beaucoup de lignes de  code, il est difficile, même pour lûauteur, de comprendre à posteriori à quoi servent les différents algorithmes.

Aujourdûhui, le programmeur travaille en **équipe**.
En effet, les programmes sont de plus en plus importants et complexes, et une personne ne peut venir à bout seule dûune application moderne.

Cûest pourquoi il est primordial de **documenter** le travail de codage.
Une partie importante de cette documentation réside dans le source même ; cûest ce quûon appelle les **commentaires**.

En Vbscipt (et également en Visual Basic) les lignes de commentaires sont précédées dûune simple quote (« ó») . Ils peuvent également être précédés par le mot « REM » (comme « remarque ») pour des raisons historiques.

Enregistrer et tester le code suivant

```
'-------------------------------------------------------
'Ceci est un commentaire
REM les commentaires commencent soit par une simple quote
'soit par REM.
'-------------------------------------------------------

'Noter le paramètre vbExclamation
MsgBox "Bonjour monde !", vbExclamation, "Ceci est le titre"
```

Fichier 003_Bonjour.vbs

Adapté dûne Bd aux Editions du Lombard, par Ridel & Mythic

LES VARIABLES

Une application qui se contenterait de « sorties écran », cêst à dire dûafficher des informations, sans faire dûopérations de calcul, serait bien pauvreé
La plupart du temps, le programmeur est appelé à manipuler ce quûon appelle des **variables**.
Celles-ci sont des lettres ou des groupes de lettres, qui permettent de stocker des valeurs **numériques** (0, -5, 2356 etc.) ou **alphanumériques** (« Bonjour », « Date : 02 mai 2001 » etc.).
« Une variable est un espace réservé correspondant à un emplacement de mémoire où vous pouvez stocker des informations susceptibles d'être modifiées pendant l'exécution du script » (Microsoft)

Il est nécessaire dûutiliser des noms de variables qui ont du **sens**, par exemple ;

AgeDuCapitaine	→ contiendrait un numérique
NomMarital	→ contiendrait une chaîne de caractères (appelé alphanumérique)
MotdePasse	→ contiendrait de lûalphanumérique.

Plus tard, nous verrons quûil est nécessaire de codifier de façon plus précise les variables, mais aujourdûhui le plus important est de leur donner du sens, et par seulement pour le créateur, mais pour toutes les personnes qui pourraient lire le source du programme.
(Les exemples qui suivent utilisent des variables qui ont peu de signification car les programmes ont un objectif technique ou pédagogique et non fonctionnel).

Enregistrer et tester le code suivant

```
'Mais combien vaut MaVariable ??
MsgBox MaVariable
```

Fichier 004_variable1.vbs

Dans le programme « Fichier 004_variable1.vbs » nous utilisons une variable (MaVariable).
Mais celle-ci ne contient aucune valeur.
Cêst pour cela que lûaffichage ne contient rien.

Enregistrer et tester le code suivant

```
MaVariable =4     'Affectation de la valeur 4 à MaVariable

MsgBox MaVariable
```

Fichier 005_variable2.vbs

Cette fois, la valeur quatre (donc cêst du numérique) a été placée dans la variable.
Nous pourrons plus tard décider dûy mettre une valeur différente (voilà pourquoi on appelle cela é une variable).

Quand une variable contient une valeur, on dit quêelle est **initialisée**.

LES DECLARATIONS

Programmer est un exercice qui est loin dêtre évident ; les pièges sont nombreux et les difficultés surgissent parfois aux moments les plus inattendues.

Le programmeur est un peu comme un bâtisseur, et si sa construction ne sâappuie pas sur des fondations solides, elle menacera de sêcrouler à tout moment.
Heureusement, un certain nombre dôoutils, de techniques et de méthodes sont là pour garantir un développement propre.

Une technique indispensable consiste à **déclarer** *explicitement* les variables **avant** de les utiliser.

Lôinstruction « option explicit », placée au début du programme, forcera la déclaration des variables.

« Vous pouvez déclarer une variable implicitement en utilisant simplement son nom quelque part dans le script. Cette pratique n'est généralement pas conseillée, puisqu'une simple faute de frappe dans un nom de variable pourrait créer des incidents imprévisibles lors de l'exécution du script. Pour éviter ce problème, l'instruction **Option Explicit** impose la déclaration explicite de toutes les variables. Cette instruction doit être la première de votre script. » (Microsoft)

Enregistrer et tester le code suivant

```
option explicit   'obligation de déclarer les variables


MaVariable =4     'Affectation de la valeur 4 à MaVariable
      '... mais MaVariable n'a pas été déclarée !!
      'Ce programme entraine une erreur
MsgBox MaVariable
```

Fichier 006_variable3.vbs

Maintenant nous mettrons systématiquement « option explicit » en début de programme, afin de pouvoir recenser rapidement les différentes variables utilisées et dêtre certain de ne pas employer une « fausse » variable qui nâaurait pas été initialisée.

Pour indiquer lôutilisation future dôune variable (pour sâautoriser à lôutiliseré), il faut utiliser lôinstruction **dim** (comme « dimensionner »).

Enregistrer et tester le code suivant

```
option explicit   'obligation de déclarer les variables
                  'C'EST UN BON PRINCIPE DE PROGRAMMATION

dim MaVariable     ' DECLARATION ! !

MaVariable =4     'Affectation de la valeur 4 à MaVariable

MsgBox MaVariable
```

Fichier 007_variable4.vbs

LES OPERATEURS

Comme tous les langages de programmation, Vbscript permet d'utiliser des **opérateurs** arithmétiques :

Opér.	Utilisation	Exemple
*	Multiplication	*result* **=** *number1****** *number2*
+	Addition	*result* **=** *number1*+*number2*
-	Soustraction	*result* **=** *number1*-*number2*
/	Division	*result* **=** *number1***/***number2*
\	Division entière	*result* **=** *number1******number2*
Mod	Effectue la division de deux nombres et renvoie seulement le reste. *result* **=** *number1* **Mod** *number2*	
^	Elève un nombre à la puissance de l'exposant indiqué. *result* **=** number**^***exposant*	

Enregistrer et tester le code suivant

```
option explicit

dim Hauteur, Largeur

Hauteur =4
Largeur =8

MsgBox Hauteur + Hauteur + Largeur + Largeur
```

Fichier 008_variable5.vbs

Créer au moins 4 petits programmes réalisant des opérations de multiplication, division é , qui seront placés dans le sous- répertoire « MesTests ».

Enregistrer et tester le code suivant

```
option explicit

 dim Hauteur, Largeur

Hauteur =4
Largeur =8

'Un peu plus d'informations s'il vous plait!
MsgBox "Le périmetre est " & ( Hauteur + Largeur ) * 2
'Le caractère & entraine une "concatènation"
```

Fichier 009_variable6.vbs

LES ENTREES / SORTIES

Jusquà présent, seules des **sorties écran** (= affichages « passif » ; pas de données entrées par lôutilisateur) ont été utilisées.

Enregistrer et tester le code suivant

```
option explicit

dim MaVar1,MaVar2

'La ligne suivante provoquera une erreur
MaVar1=inputbox "Entrer la valeur de MaVar1"
MaVar2=inputbox "Entrer la valeur de MaVar2"
'Regarder la syntaxe de "inputbox dans l'aide

MsgBox "Le resultat est " & MaVar1+MaVar2
```

Fichier 010_variable7.vbs

Enregistrer et tester le code suivant

```
option explicit

dim MaVar1,MaVar2

MaVar1=inputbox("Entrer la valeur de MaVar1")
MaVar2=inputbox("Entrer la valeur de MaVar2")

MsgBox "Le resultat est " & MaVar1+MaVar2
```

Fichier 011_variable8.vbs

Les paramètres de lôinstruction « inputbox » doivent être entre parenthèses.
Celle-ci renvoie une valeur qui est récupérée dans une variable.
Mais le résultat nôest pas satisfaisant.
Si on a tapé les chiffres 2 et 3, au lieu de trouver 5 on récupère 23é
En effet, VbScript ne connaît pas le **type de donnée** que lôon manipule et peut prendre lôinitiative de considérer ceux-ci comme du texte.
Il faut alors préciser que nous souhaitons additionner 2 entiers, comme dans le programme suivant.

Enregistrer et tester le code suivant

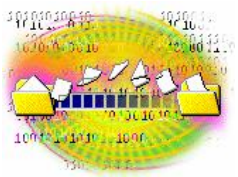
```
option explicit

dim MaVar1,MaVar2

'MaVar1 et MaVar2 sont convertis en numériques
MaVar1=Cint(inputbox ("Entrer la valeur de MaVar1"))
MaVar2=Cint(inputbox ("Entrer la valeur de MaVar2"))

MsgBox "Le resultat est " & MaVar1 + MaVar2
```

Fichier 012_variable9.vbs

DISCUSSION :
Avec des langages plus « typés » (qui obligent à préciser le type des données de façon plus fine), les variables MaVar1 et MaVar2 auraient été déclarées comme des entiers, et il aurait été inutile d'utiliser une fonction comme « cint ».

Avez-vous examiné les rubriques d'aide concernant « InputBox » et « cint » ?
Si ce n'est pas le cas, astreignez-vous, dès la rencontre d'un nouveau mot, à utiliser cette démarche, car elle reste permanente dans la vie d'un programmeur, du débutant au plus expérimenté.

LES BOUCLES



« L'existence n'est qu'une succession ininterrompue d'événements passés » Nietzsche

Les boucles (ou itérations) sont un des fondements de la programmation.
Elles permettent de reproduire une suite d'instructions.

Exemples de boucles :
« Tant que l'utilisateur n'a pas entré une valeur supérieure à zéro, redemander la valeur »
« Pour tous les nombres compris entre 10 et 15, afficher ceux qui sont divisibles par 2 »
« En partant de 'A', aller jusqu'à 'Z' et afficher la lettre »

Enregistrer et tester le code suivant

```
option explicit

dim i

for i=10 to 15
      MsgBox i
next
```

Fichier 013_boucle1.vbs

COMMENTAIRES :
for i = 10 to 15 signifie : assigner la valeur 10 à la variable i
next signifie : incrémenter (augmenter) la variable i et repartir sur la ligne suivant le ***for***.

Lorsque la variable i aura dépassé la valeur 15, alors le programme sortira de la boucle et continuera les instructions situées après le ***next***.

Enregistrer et tester le code suivant

```
option explicit

dim i

for i=10 to 15 step 2
      MsgBox i
next
```

Fichier 014_boucle2.vbs

COMMENTAIRES :
Cette fois, le « pas » des itérations sera de 2.
Si ce n'est pas déjà fait, regarder l'aide concernant « For...Next »

Exercice 1 :

Réaliser un programme qui effectue une boucle de 1 à 5, qui additionne au fur et à mesure le chiffre du compteur et affiche le résultat.
1ère Itération => affichage de **1**
2ème Itération => affichage de **3** *(1 + 2)*
3ème Itération => affichage de **6** *(1 + 2 + 3)*
4ème Itération => affichage de **10** *(1 + 2 + 3 + 4)*
5ème Itération => affichage de **15** *(1 + 2 + 3 + 4 + 5)*

Fichier 015_boucle3.vbs

Exercice 2 :

Compléter le code suivant afin que le programme affiche une liste de chiffres dont les bornes basses, hautes et l'intervalle (le pas) sont entrées par l'utilisateur.

```
option explicit

dim i,BorneDebut,BorneFin,Pas

BorneDebut = inputbox("Entrer la valeur de début de boucle")
BorneFin   = inputbox("Entrer la valeur de fin de boucle")
Pas        = inputbox("Entrer le pas")
```

Fichier 016_boucle4.vbs

Exercice 3 :

Sachant que « msgbox chr(64) » affiche la lettre A, « msgbox chr(65) » affiche la lettre B etc. , afficher la liste de toutes les lettres de l'alphabet.

1ère Itération => affichage de **A**
2ème Itération => affichage de **B**
é
Dernière itération => affichage de **Z**

Fichier 017_boucle5.vbs

Solution de l'exercice 1 :


```
option explicit

dim i,j

for i = 1 to 5
      j=j+i
      MsgBox j
next
```

Solution de l'exercice 2 :

```
option explicit

dim i,BorneDebut,BorneFin,Pas

BorneDebut = inputbox("Entrer la valeur de début de boucle")
BorneFin   = inputbox("Entrer la valeur de fin de boucle")
Pas        = inputbox("Entrer le pas")

for i = BorneDebut to BorneFin step Pas
      MsgBox i
next
```

Que se passe t-il si on entre comme borne début 20, borne fin 10 avec un pas de ï 2 ?

Solution de l'exercice 3 :

```
option explicit

dim i

for i = 65 to 90
      MsgBox chr(i)
next
```

Complément :

Enregistrer et tester le code suivant

```
option explicit
dim i,j
for i = 65 to 90
        j =j & chr(i)
next
msgbox j
```

018_boucle6.vbs

Analyser et commenter le code.

(é . Avez-vous consulté lôaide sur la fonction CHR ?é .)

La boucle est un instrument extrêmement puissant.
Son utilisation dans les programmes est quasi systématique, mais là ou elle fait « parler la poudre », cœst lorsquœelle est « imbriquée ».
Dans ce cas, une boucle contient elle-même une autre boucle.

Enregistrer et tester le code suivant

```
option explicit

dim i,j

for i = 1 to 4
      for j = 100 to 102
            msgbox "i=" & i & "  j=" & j
      next
next
```

Noter bien que pour chaque « for » on doit trouver un « next ».

Fichier 019_boucle7.vbs

Enregistrer et tester le code suivant

```
option explicit

dim i,j,k, compteur

for i = 1 to 2
      for j = 1 to 2
            for k = 1 to 2
                  msgbox i & j & k
                  compteur=compteur+1
            next
      next
next

msgbox compteur & " itérations !"
```

Fichier 020_boucle8.vbs

EXERCICES :
Rechercher lôinstruction While...Wend dans lôaide.
Refaire les exercices 1, 2 et 3 en utilisant lôinstruction While...Wend.

Rechercher « Do...Loop » dans lôaide.

LES CONDITIONS

(appelées aussi « structures alternatives »)

Lorsqu'un traitement doit être subordonné à une condition, on utilise la démarche :

```
if (EXPRESSION_BOOLEENNE)
        INSTRUCTION
fin if
```

Une EXPRESSION_BOOLEENNE est une expression (c.a.d un mot réservé, une fonction, un objet, une choseé) qui renvoi VRAI ou FAUX.
En principe (cela dépends des langages) VRAI est positif et FAUX est inférieur ou égal à zéro.
Exemple :

```
X=4
si ( X > 1 ) alors
        afficher « X est superieur à 1 »
fin si
```

Lôexpression (x > 1) renvoi VRAI puisque 4 est supérieur à 1.

Enregistrer et tester le code suivant

```
'------------------------------------------------------------
'resolution d'une equ. du second degre a coeff. non nuls
'------------------------------------------------------------

option explicit

dim A, B, C, Delta

'  acquisition
A=inputbox("entrez la valeur du coefficient A ( non nul )")
B=inputbox("entrez la valeur du coefficient B ( non nul )")
C=inputbox("entrez la valeur du coefficient C ( non nul )")

' traitement et affichage des resultats
Delta = B * B - 4 * A * C

if Delta = 0 then msgbox "X1 = X2 = " & ( -B / (2 * A))

if Delta > 0 then
    msgbox  "X1 = " & (-B + sqrt(Delta)) / (2 * A)
    msgbox  "X2 = " & (-B - sqrt(Delta)) / (2 * A)
end if

  if Delta < 0 then msgbox "pas de racines reelles"
```

Fichier 021_condition1.vbs

Voici le même programme en C++ :

```
#include<math.h>
int main()
 {
 double A, B, C, Delta;
 /* acquisition */
 cout <<  "entrez les valeurs des coefficients A, B, et C non nuls";
 cout << endl;
 cin >> A >> B >> C;
 /* traitement et affichage des resultats */
 Delta = B * B - 4 * A * C;
 if (Delta == 0) cout << "X1 = X2 = " << -B / (2 * A) << endl;
 if (Delta > 0)
   {
   cout <<  "X1 = " << (-B + sqrt(Delta)) / (2 * A) << endl;
   cout <<  "X2 = " << (-B - sqrt(Delta)) / (2 * A) << endl;
   }
 if (Delta < 0) cout << "pas de racines reelles" << endl;
 }
```

Observez bien les 2 programmes.
En C++, « cout » est lôéquivalent du msgbox et « cin » lôéquivalent du inputbox.

Quelques éléments de syntaxe sont différents, mais la logique est la même !

Lorsque le choix entre deux traitements est subordonné à une condition, on utilise la structure :

```
if (EXPRESSION_BOOLEENNE)
        INSTRUCTION 1
sinon
     INSTRUCTION 2
fin if
```

Par exemple :

```
If x > 0 then
        Message = « Superieur à zéro »
Else
        Message = « Inférieur ou égal à zéro »
End if
Msgbox Message
```

Il est possible dôaller encore plus loin dans les imbrications de conditions avec **elseif**.

Lorquôon maîtrise lôutilisation les boucles et les conditions, on couvre déjà une bonne partie des besoins en programmation.
Un besoin classique est de sôassurer quôune saisie est correcte :

Enregistrer et tester le code suivant

```
option explicit

dim NombreSaisi

NombreSaisi=0

while NombreSaisi < 10
  NombreSaisi = inputbox("Entrez un nombre superieur à 10")
  if NombreSaisi < 10 then msgbox "SUPERIEUR A 10 S.V.P !"
wend
```

Fichier 022_condition2.vbs

Exercice 4 :
Réaliser un programme qui demande une saisie entre 10 et 20 ; si la saisie est en dehors des bornes, affichage dôun message puis on boucle tant que la saisie est mauvaise.

Exercice 5 :
Réaliser un programme qui demande de saisir un âge.
Si la saisie est inférieure à 1, on recommence la saisie.
Si lôâge est inférieur à 18 ans, affichage de « vous êtes mineur ».
Si lôâge est supérieur à 18 ans, affichage de « vous êtes majeur »

Solution exercice 4

```
option explicit

dim NombreSaisi

NombreSaisi=0

while NombreSaisi < 10
  NombreSaisi = inputbox("Entrez un nombre entre 10 et 20")
  if NombreSaisi < 10 then msgbox "entre 10 et 20 S.V.P !"
wend
while NombreSaisi > 20
  NombreSaisi = inputbox("Entrez un nombre entre 10 et 20")
  if NombreSaisi > 20 then msgbox "entre 10 et 20 S.V.P !"
wend
```

Fichier 023_condition3.vbs

Solution exercice 5

```
option explicit

dim UnAge

UnAge=0

while UnAge < 1
  UnAge = inputbox("Entrez votre age")
  if UnAge > 18 then
      msgbox "vous etes majeur"
  elseif UnAge > 0 then
      msgbox "vous etes mineur"
  end if
wend
```

Fichier 024_condition4.vbs

Question : Comment se comporte le programme si on entre 18 ?
Remplacer le signe > par >= (superieur ou égal) et tester la différence.

Rechercher et étudier « Opérateurs de comparaison » dans l'aide

Une structure alternative pratique est « select é case »
Elle permet de traiter beaucoup de tests conditionnels en une seule fois.

Enregistrer et tester le code suivant

```
option explicit

dim UnChiffre

UnChiffre = inputbox("Entrer un chiffre")

Select Case UnChiffre
Case 1,3,5,7,9
  MsgBox "Nombre premier"
Case 2,4,6,8
  MsgBox "Pas un nombre premier"
Case Else
  MsgBox "J'analyse seulement entre 1 et 9"
End Select
```

Fichier 025_condition5.vbs

Exercice 6 :
Réaliser une mini calculette.
Ce programme demande un nombre, puis un opérateur (+,-,*,/), puis un autre nombre et affiche le résultat de lôopération.
Le programme affiche ensuite « Voulez-vous continuer (o/n)? ».
Tant que lôutilisateur tape « o », le programme recommence.

AIDE via pseudo-code :

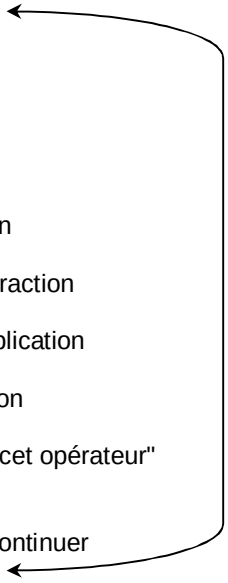
'Declarations
Variable Continuer
Variables Nombre1,Nombre2,Operateur

'Initialisation
Continuer = "o"

Tant que Continuer = "o"
  Demander Nombre1
  Demander Operateur
  Demander Nombre2

  Selon lôopérateur choisis
  Si "+"
    Afficher le résultat de lôaddition
  Si "-"
    Afficher le résultat de la soustraction
  Si "*"
    Afficher le résultat de la multiplication
  Si "/"
    Afficher le résultat de la division
  Sinon
   Afficher "Je ne reconnaît pas cet opérateur"
  Fin selon

  Demander si lôutilisateur veut continuer
Fin du Tant que Continuer



Solution exercice 6


```
option explicit

'Declarations
dim Continuer
dim Nombre1,Nombre2,Operateur

'Initialisation
Continuer = "o"

while Continuer = "o"
 Nombre1   = inputbox("Entrer un nombre")
 Operateur = inputbox("Entrer un opérateur ( +,-, / , * )")
 Nombre2   = inputbox("Entrer un autre nombre")

 Select Case Operateur
 Case "+"
  'Conversion due à la concaténation par défaut
  msgbox cint(Nombre1) + cint(Nombre2 )
 Case "-"
  msgbox Nombre1 - Nombre2
 Case "*"
  msgbox Nombre1 * Nombre2
 Case "/"
  msgbox Nombre1 / Nombre2
 Case Else
  MsgBox "Je ne reconnait pas cet opérateur"
 End Select

 Continuer=inputbox("Voulez-vous continuer (o/n)?")
wend
```

Fichier 026_condition6.vbs

OPERATEURS LOGIQUES

Les opérateurs AND (ET) et OR (OU) deviennent vite indispensables, dès lors que l'on utilise les conditions.

L'opérateur AND oblige les deux expressions (*result* **=** *expression1* **And** *expression2)* à être vraies pour que l'ensemble soit VRAI.

Par exemple :
Si (X est divisible par 3) AND (X est divisible par 2) alors
Traitement
FIN SI
Dans le cas ou X est égal à 6, alors les deux conditions sont vraies et le traitement est effectué.
Si X est égal à 4, alors le traitement ne sera pas effectué car une des 2 conditions n'est pas vraie.

Avec le OR
Si (X est divisible par 3) OR (X est divisible par 2) alors
Traitement
FIN SI
Dans le cas ou X est égal à 6, alors les deux conditions sont vraies et le traitement est effectué.
Si X est égal à 4, alors le traitement sera **aussi** effectué car **une des 2 conditions est vraie**, et cela suffit.
Pour que le traitement ne soit pas effectué, il faut que les deux conditions soient fausses (si X=5é).

Enregistrer et tester le code suivant

```
option explicit

dim UnChiffre

UnChiffre = inputbox("Entrer un chiffre")

if UnChiffre=1 OR UnChiffre=3 OR UnChiffre=5 then
      MsgBox "Chiffre inpair"
end if

if UnChiffre=2 OR UnChiffre=4 then
      MsgBox "Chiffre pair"
end if

if UnChiffre < 1 OR UnChiffre > 5 then
      MsgBox "J'analyse seulement entre 1 et 5"
end if
```

Fichier 027_and_or1.vbs

EXERCICES DE SYNTHESE

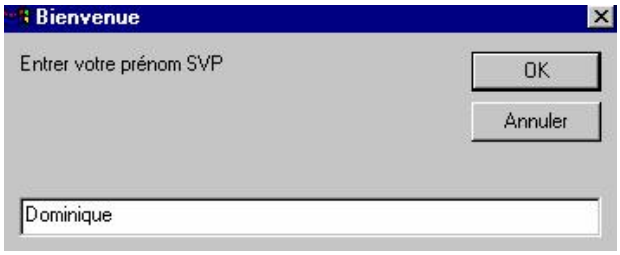
Exercice « Trajet »

(fichier 028_synthese1.VBS - solution hors support -)

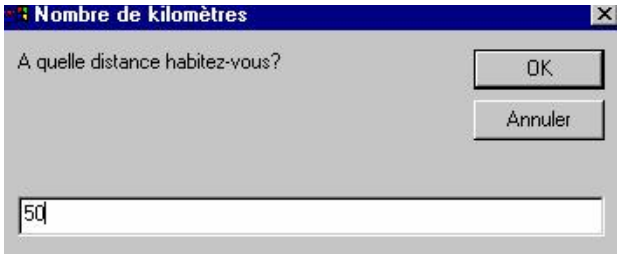
Faire un programme qui

- demande le prénom de l'utilisateur.
- demande « A quelle distance habitez-vous (en km) ?»
- Combien de temps mettez-vous pour venir (en mn)?
- Affiche « [*prénom*], vous roulez en moyenne à [*xx*] km/h
- Si la moyenne est supérieure à 60 :
  - Affiche « Soyez prudent »
- Si la moyenne est entre 40 et 60 :
  - Affiche « C'est raisonnable »
- Si la moyenne est inférieure à 40 :
  - Affiche « Faites réviser votre véhicule »

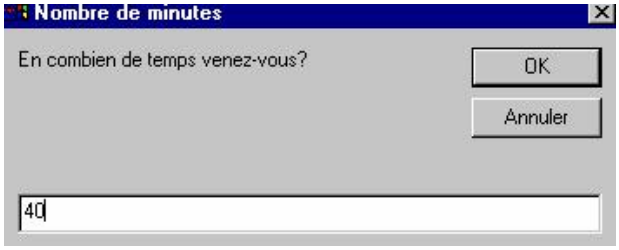
1



2



3



4

EN RESPECTANT SCRUPULEUSEMENT LES INTERFACES PROPOSEES CI-DESSUS

Exercice « Alphabet »

(fichier 029_synthese2.VBS - solution hors support -)

Faire un programme qui demande une lettre et qui affiche :
« Cette lettre est en [*xx*] position dans l'alphabet. »

par exemple : D donnera → *cette lettre est en position 4*

Le programme fonctionnera correctement avec des lettres majuscules ou minuscules.

SOUS-PROGRAMMES (fonctions et procédures)

Les *sous-programmes* (fonctions et procédures) offrent au programmeur le moyen de réaliser des *modules* de programmation relativement indépendants, réalisant chacun une tâche parfaitement définie, et pouvant être développés, vérifiés et testés séparément.

Une fois écrit et mis au point, un sous-programme pourra être vu et utilisé comme une « boîte noire », dont on aura à connaître que le nom (symbolisant le calcul réalisé), et les paramètres d'entrée et de sortie.
L'exécution d'une fonction, ou plus généralement d'un sous-programme, est provoquée par son **appel** qui figure dans le programme principal (ou dans un autre sous-programme). L'appel provoque un *débranchement* temporaire de l'exécution du programme principal (ou du sous-programme) *appelant* , qui est interrompu pour laisser s'exécuter le sous-programme *appelé* . Après la fin de l'exécution du sous-programme appelé, l'exécution du programme appelant reprend là où elle avait été interrompue.

Dans VBScript (comme dans la plupart des langages), il existe deux types de sous-programmes: les SUB et les FUNCTION (Fonctions).

Procédures ; mot clef **Sub**.
Une procédure Sub est une série d'instructions VBScript, encadrée par les instructions Sub et End Sub, qui effectue des actions mais ne renvoie pas de valeur. Une procédure Sub peut accepter des arguments (constantes, variables ou expressions transmises par une procédure appelante). Si une procédure Sub n'a pas d'arguments, son instruction Sub doit présenter une paire de parenthèses vide.

Exemple d'une procédure sans arguments :
Sub AfficheBonjour()
Msgbox « Bonjour »
End Sub

Exemple d'appel :

option explicit

Sub AfficheBonjour()
Msgbox " Bonjour "
End Sub

'Le programme commence ici
AfficheBonjour
'Le programme s'arrête là

La procédure est construite ici au début du programme, puis utilisée par la suite.
Il est plus courant de placer les sous-programmes vers la fin du source :
option explicit

'Le programme commence ici
AfficheBonjour
'Le programme s'arrête là

Sub AfficheBonjour()
Msgbox " Bonjour "
End Sub

Exemple d'une procédure avec arguments en entrée :
Sub AfficheBonjour(Titre)
Msgbox « Bonjour »,,Titre
End Sub

Exemple d'appel :
'
'
AfficheBonjour(« Bienvenue »)
'
'

Enregistrer et tester le code suivant

```
option explicit

RepeteLettre_A 5

Sub RepeteLettre_A(NbLettres)
' Variables déclarées en local
dim i, Msg
  for i=1 to NbLettres
      Msg=Msg & "A"
  Next
msgbox Msg
End Sub
```

Fichier 030_sous_prog1.vbs

Dans l'exemple précédant, les variables i et Msg sont déclarées dans la procédure.
Elles ne seront pas *visibles* (donc pas connues) par le reste du programme
Si vous essayez le code suivant :

```
option explicit

RepeteLettre_A 5

msgbox Msg

Sub RepeteLettre_A(NbLettres)
dim i, Msg
  for i=1 to NbLettres
    Msg=Msg & "A"
  next
End Sub
```

VBScript enverra une erreur sur la ligne 5, car la variable n'est connue que *localement* dans la procédure.

Nous aborderons plus loin les concepts de variables *locales* et *globales*.

Exercice N°8 Fichier 031_sous_prog2.vbs

Faire un programme qui demande une lettre, le nombre de répétitions de celle-ci et affiche le résultat.
Le programme contiendra une procédure qui acceptera deux paramètres ; la lettre à répéter et le nombre de répétitions.

Exercice N°9 Fichier 032_sous_prog3.vbs

Coder un programme qui exploite deux procédures :

① Une procédure qui demande un age entre 10 et 20 ; si la saisie est en dehors des bornes, affichage d'un message puis on boucle tant que la saisie est mauvaise (voir Exercice 4)

② Une autre procédure qui analyse le nombre saisi (voir Exercice 5).

Si celui-ci est inférieur à 18 ans, affichage de « vous êtes mineur ».
Si celui-ci est supérieur à 18 ans, affichage de « vous êtes majeur »

Solution de l'exercice N°8

```
option explicit

dim UneLettre
dim NbRepet

UneLettre=inputbox ("Entrer une lettre")
NbRepet=inputbox ("Entrer le nombre de répétitions")

RepeteLettre  UneLettre, NbRepet

Sub RepeteLettre( p_UneLettre, p_NbRepet )
dim i, Msg
  for i=1 to p_NbRepet
      Msg=Msg & p_UneLettre
  Next
msgbox Msg
End Sub
```

Solution de l'exercice N°9

```
option explicit

dim UnAge

DemandeAge

Analyse

'----------------------------
'Fin du programme
'----------------------------
'----------------------------
'Déclarations des procedures
'----------------------------

sub DemandeAge()

UnAge=0

while UnAge < 10
  UnAge = inputbox("Entrez un nombre entre 10 et 20")
  if UnAge < 10 then msgbox "entre 10 et 20 S.V.P !"
wend
while UnAge > 20
  UnAge = inputbox("Entrez un nombre entre 10 et 20")
  if UnAge > 20 then msgbox "entre 10 et 20 S.V.P !"
wend

end sub

'-----------------------------------------------------
sub Analyse()

 if UnAge > 18 then
      msgbox "vous etes majeur"
 elseif UnAge > 0 then
      msgbox "vous etes mineur"
 end if

end sub
```

Fonctions **; mot clef** Function**.**

Une procédure **Function** est une série d'instructions VBScript encadrée par les instructions **Function** et **End Function**.
Une procédure **Function** est semblable à une procédure **Sub** mais peut renvoyer une valeur.

Enregistrer et tester le code suivant

```
option explicit

msgbox MettreAuCarre(9)

'------------------------
'Fin du programme
'------------------------

Function MettreAuCarre(UneValeur)
   MettreAuCarre = UneValeur * UneValeur
End Function
```

Fichier 033_sous_prog4.vbs

Principe : Pour quûne fonction retourne une valeur, il faut que dans son **corps** le nom de la fonction soit affecté.
Syntaxe :
Function NOM_DE_LA_FONCTION (Parametre1, Parametre2 é)
Traitement é .
Traitement é .
Traitement é .
NOM_DE_LA_FONCTION = UneValeur
End Function

Dans lûexemple précédant, le résultat de la fonction « MettreAuCarre » est directement exploité par lûinstruction msgbox.
Selon le même principe, son résultat peut-être directement utilisé comme paramètre dûune autre fonctioné

Enregistrer et tester le code suivant

```
option explicit

msgbox Ajoute1(MettreAuCarre(9))

'------------------------
'Fin du programme
'------------------------

Function MettreAuCarre(UneValeur)
   MettreAuCarre = UneValeur * UneValeur
End Function

Function Ajoute1(UneValeur)
   Ajoute1 = UneValeur + 1
End Function
```

Résultat = 82

Fichier 034_sous_prog5.vbs

Portée et durée de vie des variables

« Lorsque vous déclarez une variable dans une procédure, seul le code de cette procédure peut accéder à la valeur de cette variable et la modifier; elle a une portée *locale* et est qualifiée de variable de *niveau procédure*. Cependant, vous devez parfois utiliser une variable d'une plus large portée, notamment lorsque sa valeur doit être accessible par toutes les procédures d'un script. Si vous déclarez une variable en dehors d'une procédure, toutes les procédures du script peuvent la reconnaître. Ce type de variable *de niveau script* a ce qu'il est convenu d'appeler une portée de niveau script.

La *durée de vie* d'une variable est comprise entre la déclaration de la variable et la fin de l'exécution du script. Celle d'une variable locale va de sa déclaration à la fin de la procédure. Les variables locales constituent un espace de stockage temporaire idéal pendant l'exécution d'une procédure. Plusieurs variables locales peuvent porter le même nom dans différentes procédures, puisque chacune est reconnue uniquement par la procédure dans laquelle elle est déclarée.

La portée d'une variable est fonction de l'endroit où vous la déclarez. Au niveau script, la durée de vie d'une variable est toujours la même. Elle existe tant que le script s'exécute. Au niveau procédure, cette durée est limitée à l'exécution de la procédure. La variable est détruite automatiquement à la sortie de la procédure. »

(© Microsoft)

" *Le battement d'une aile d'un papillon au Brésil peut déclencher une tornade au Texas*. " Edward **Lorenz**

TECHNIQUES DE DEBUGAGE

Que faire lorsque le programme ne marche pas ?
Cela peut venir d'un problème de syntaxe ou d'un problème d'algorithme.

Voici trois techniques qui ont fait leurs preuves depuis que l'informaticien est en prise avec des lignes de code.
Ce sont des règles d'or qui vous sauveront la mise dans beaucoup de situations.

Règle N° 1 : Informer

Placer à des endroits judicieux des messages qui permettront de connaître le contenu des variables.

Règle N° 2 : Simplifier

Tant pis si le code est redondant ; la priorité est que ça fonctionne.

Trouver un moyen différent de faire la même chose ; il existe toujours une solution plus simple (même si elle est moins élégante).

Règle N° 3 : Isoler

Créer des petits programmes de test qui reprennent une partie critique du code.

La dernière extrémité, mais qui marche toujours ; enlever la moitié du source, tester/debugger puis recommencer l'opération si le bug n'est pas levé.

Un programme répond à un besoin.
Celui-ci est formalisé par un utilisateur (ou son représentant).
Son expression peu prendre milles formes différentes ;

- Le résultat dûun interview
  - Fait par un commercial
  - Fait par un chef de projet
  - Fait par un développeur
  - é
- Un existant à modifier
- Un dossier plus ou moins complet respectant une méthode
- Etc.

On parle souvent de Cahier des charges. (CDC).
Cette appellation est fréquemment utilisée à tort et à travers, et peut se présenter sous la forme dûune feuille simple, dûun brouillon papieré et dans lûidéal dûun ou plusieurs dossiers contenants lûidée générale jusquûaux descriptions des routines internes en passant par un planning et un plan qualité.

REALISATION DûUN PROGRAMME A PARTIR DûUN RAPPORT.

Projet « Le piquet à cheval »
Voici notre « cahier des charges :

Decremps, dans son Codicille de Jérome Sharp (1783, in-8°) nous explique un jeu de calcul.
« J'allais un jour à la campagne avec un de mes amis, et nous étions tous les deux à cheval. Il me proposa de jouer au piquet et je lui répondis que je jouerai volontiers une partie quand nous serions arrivés.
Mais, me dit-il, nous pouvons jouer au piquet sans carte et sans mettre pied à terre.
Comme je ne connaissais pas le jeu qu'il me proposait il me l'expliqua en me disant qu'un de nous prendrait à volonté un nombre quelconque depuis un jusqu'à dix ; que l'autre y ajouterait un autre nombre pris également dans la dizaine pour en avoir la somme ; que le premier ajouterait à cette somme tel nombre qu'il voudrait, pourvu que ce fût toujours au-dessous de onze, et que celui de nous qui, en ajoutant ainsi alternativement, arriverait le premier à cent, gagnerait la partie.

Les règles de ce jeu me parurent bien simples ; je nommai premièrement 5 ; il ajouta 10, pour avoir 15 ; j'ajoutai 10 pour avoir 25 ; il ajouta 5 pour faire 30 ; je nommai 1 pour 31 ; et lui 7 pour 38 ; moi 9 pour 47, et lui 9 pour 56 ; moi 4 pour 60 et lui 7 pour 67 ; moi 3 pour 70 et lui 8 pour 78 ; moi 2 pour 80 et lui 9 pour 89.
Dès ce moment, je compris, sans finir la partie, que j'avais perdu ; car, dis-je, en moi-même, si j'ajoute 1 pour 90, il ajoutera 10 pour faire 100, et si j'ajoute 10 pour 99, il aura 100 en ajoutant 1 ; en un mot, quelque nombre que je choisisse, il n'aura qu'à ajouter ce qui manque pour finir la partie et la gagner.

J'observai donc que l'essentiel consistait à s'emparer du nombre 89 ; je demandai ma revanche, mais mon adversaire arriva le premier à 78, et je m'aperçus alors que j'aurais autant de difficultés à attraper 89 que j'en avais eu auparavant à attraper le nombre 100.
Je commençai une troisième partie en me proposant de parvenir moi-même le premier au nombre 78, pour passer de là à 89, et puis à 100 ; mais dans cette autre partie, mon adversaire arriva le premier au nombre 67 ; j'y ajoutai 1 pour 68, et lui 10 pour 78. Je m'aperçus alors que mon adversaire avait une marche sûre et je m'appliquai à la trouver.
Je découvris, en y réfléchissant, que les numéros dont il fallait s'emparer pour être sûr de gagner, étaient ceux-ci, pris dans un ordre rétrograde :

89, 78, 67, 56, 45, 34, 23, 12, 1.

Réfléchissant ensuite sur la nature de ce jeu, je fis des découvertes qui me servirent à gagner ma revanche.

J'observai d'abord que les nombres ci-dessus, 1, 12, 24, 33, etc., pris dans leur ordre naturel, forment une progression arithmétique dont la différence est 11, c'est à dire que chaque terme surpasse celui qui le précède de ce nombre 11. Je remarquai ensuite que ces mêmes nombres dépassent chacun d'une unité seulement les nombres suivants, composés chacun des deux chiffres semblables :

11, 22, 33, 44, 55, 66, 77, 88, 99.

Cette dernière remarque me parut utile pour soulager la mémoire.

Quand je connus la marche générale et le moyen de gagner dans tous les cas, je demandai ma revanche.

Mon adversaire, qui ne soupçonnait pas la découverte que je venais de faire, souscrivit à ma proposition, et comme il me permit, en commençant la partie, de m'emparer des nombres 12, 23, 34, espérant que je ne suivrais point la progression qu'il croyait m'être inconnue, il se trouva frustré de son espérance et compris bien que j'avais découvert son secret. »

- TP - Objectif:

Réaliser un programme interactif qui luttera contre
un adversaire "humain".

La programmation en Vbscript peu sâavérer délicate lorsque le programme est conséquent, aussi il convient de partir sur une analyse qui découpe le processus de développement en plusieurs étapes.

Recommandations :
Analyser dâabord sur papier en imaginant le programme avec ses différentes parties.

Si vous avez des difficultés, vous trouverez page suivante deux approches de construction.

Démarches de codage proposée:

☒ Démarche 1 : « Petit à petit, l'oiseau é » (Fichier 035_piquet1.vbs - *solution hors support*-)

Réaliser les cinq étapes qui suivent en les enregistrant à chaque fois dans des fichiers différents (exemple ; piquet1, piquet2 é).
Ainsi, vous ne passerez à l'étape suivante qu'en étant certain que le code précédant fonctionne correctement.

1) Réaliser une boucle infinie
qui demande un nombre <=10 et contrôle cette limite.

2) Additionner ce nombre et afficher le total au fur et
a mesure des saisies

3) Ajouter un choix au début: "qui commence?" puis une fois
sur deux faire ajouter 5 au total et afficher
"VbScript ajoute 5 ".

4) Si le total atteint ou dépasse 100 :
    si c'est au programme de jouer
        afficher "Bravo, vous avez gagné..."
    sinon
        afficher "Désolé, vous avez perdu !"
  terminer le programme

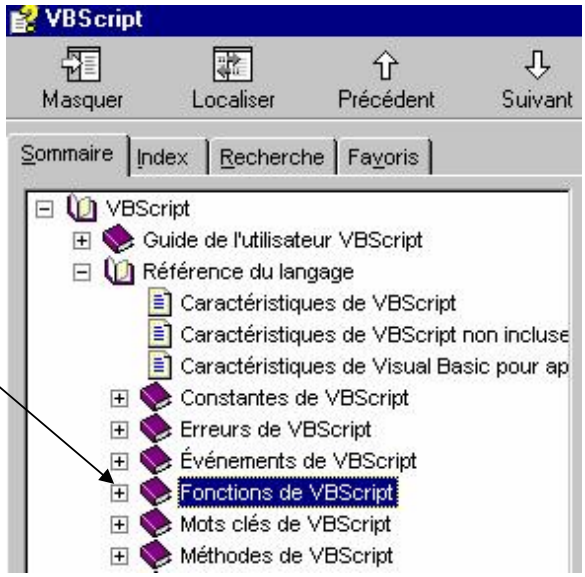
5) Concernant le jeu machine, le mettre en place en
  utilisant les bornes 1, 12, 23, 34, 45, 56, 67, 78, 89 telles
  que décrites dans le support papier

☒ Démarche 2 : « Diviser pour mieux régner » (Fichier 036_piquet2.vbs - *solution hors support*-)
Le programme est découpé en procédures (majuscules dans cet exemple)

```
INITIALISATIONS

CHOIX_JOUEUR_COMMENCER

While Total < 100
  if Joueur = JoueurHumain then JEU_HUMAIN
  if Total < 100 then JEU_MACHINE
Wend

ANNONCE_VAINQUEUR

'-------------------------------------------------------
'FIN DU PROGRAMME                    -
'-------------------------------------------------------

'-------------------------------------------------
'DECLARATIONS DES ROUTINES -
'-------------------------------------------------
é
é
```

LES OUTILS DU LANGAGE

Les langages de programmation fournissent un ensembles de fonctions qui facilitent grandement le travail du développeur.
Il n'est pas nécessaire de toutes les connaître par cœur.
En effet, toutes ne sont pas utilisées aussi fréquemment.
Cependant, il est utile de les parcourir attentivement (voir le fichier d'aide) **au moins une fois**.



LISTE DES FONCTIONS VBSCRIPT

Abs
Array
Asc
Atn
CBool
CByte
CCur
CDate
CDbl
Chr
CInt
CLng
Cos
CreateObject
CSng
CStr
Date
DateAdd
DateDiff
DatePart
DateSerial
DateValue
Day
Eval
Exp
Fix
FormatCurrency
FormatDateTime
FormatNumber
FormatPercent
GetLocale
GetObject
GetRef
Hex
Hour
InputBox
InStr
InStrRev
Int
IsArray
IsDate
IsEmpty
IsNull
IsNumeric
IsObject
Join
LBound
LCase
Left
LoadPicture
Log
LTrim
Mid
Minute
Month
MonthName
MsgBox
Now
Oct
Replace
RGB
Right
Rnd
Round
RTrim
ScriptEngine
ScriptEngineBuildVersion
ScriptEngineMajorVersion
ScriptEngineMinorVersion
Second
Sgn
Sin
Space
Split
Sqr
StrComp
String
StrReverse
Tan
Time
Timer
TimeSerial
TimeValue
Trim
TypeName
UBound
UCase
VarType
Weekday
WeekdayName
Year

Les fonctions **Mid** et **Len**

Ces fonctions sont très utilisées. Regarder leur explication dans l'aide.

Enregistrer et tester le code suivant

```
option explicit

dim UneChaine
dim debut, longueur

UneChaine = Inputbox("Entrer une chaîne de caractères")
debut     = Inputbox("Position ", "Début de l'extraction")
longueur  = Inputbox("Longueur", "Fin de l'extraction")

msgbox mid(UneChaine,debut,longueur)
```

Fichier 037_fctMid1.vbs

Voici un programme qui compte le nombre de fois ou la lettre « a » est trouvée dans une chaîne.

Enregistrer et tester le code suivant

```
option explicit

dim UneChaine
dim i, Compteur_a

Compteur_a=0

UneChaine = Inputbox("Entrer une chaine de caractères")

for i=1 to len(UneChaine)
  if mid(UneChaine,i,1)="a" then
      Compteur_a = Compteur_a + 1
  end if
next

msgbox "nombre de ""a"" trouvés:" & Compteur_a
```

Fichier 038_fctMid2.vbs

NB : Pour afficher en sortie (via la MsgBox) le caractère « a » entre guillemet, il est nécessaire de doubler les guillemets, afin qu'ils ne soient pas interprétés. Cette astuce se retrouve dans de nombreux langages.

Exercice 10

Réaliser un programme qui affiche successivement tous les mots d'une phrase entrée au clavier.

Fichier 039_fctMid3.vbs

Exercice 11

Réaliser un programme qui inverse une phrase entrée au clavier.
(exemple : abcdef deviendra fedcba)

Fichier 040_fctMid4.vbs

Solution de l'exercice N°10

```
option explicit
'Compte le nombre de mots d'une chaine

dim UneChaine, UnMot
dim i


UneChaine = Inputbox("Entrer une chaine de caractères")

for i=1 to len(UneChaine)
  if mid(UneChaine,i,1)<>" " then
      UnMot=UnMot & mid(UneChaine,i,1)
  Else
      'Trim enlève les blancs
      if len(Trim(UnMot))>0 then msgbox UnMot
      UnMot=""
  end if
next

'Pour le dernier mot
if len(Trim(UnMot))>0 then msgbox UnMot
```

Fichier 039_fctMid3.vbs

Solution de l'exercice N°11

```
option explicit
'Inverse une chaine de caractères

dim UneChaine, UnMot
dim i


UneChaine = Inputbox("Entrer une chaine de caractères")

for i=len(UneChaine) to 1 step -1
      UnMot=UnMot & mid(UneChaine,i,1)
Next

msgbox UnMot
```

Fichier 040_fctMid4.vbs

Pour la petite histoire , un palindrome est une phrase, un nombre, un message, qui peut être lu de droite à gauche ou de gauche à droite en gardant le même sens ou plutôt la même signification.

Voici un exemple édifiant :

Leon a rasé césar a noel

Et un autre surprenant :

1367631 = 111 * 111 * 111 = 1367631

La fonction **Timer**

Celle-ci Renvoie le nombre de secondes qui se sont écoulées depuis minuit.
Son utilité n'est pas évidente à priori ; cependant elle est très pratique.
Par exemple, dans le programme du « Piquet à cheval », nous aurions pu l'utiliser pour générer un chiffre au hasard (ou pseudo hasard).
Une autre utilisation intéressante est de s'en servir pour savoir combien de temps prends un programme (ou une portion de celui-ci) afin d'améliorer ses performances.

L'exemple suivant utilise la fonction **Timer** pour déterminer le temps nécessaire pour l'itération d'une boucle **For...Next** un nombre *N* spécifié :

Enregistrer et tester le code suivant

```
option explicit


msgbox CombienTps(inputbox ( "Entrer un nombre" ))

Function CombienTps(N)
   Dim StartTime, EndTime, I
   StartTime = Timer
   For I = 1 To N
   Next
   EndTime = Timer
   CombienTps = EndTime - StartTime
End Function
```

Fichier 041_fctTimer1.vbs

Utilisons cette fonctionnalité pour essayer d'améliorer une routine :

Enregistrer et tester le code suivant

```
option explicit

Dim StartTime

StartTime = Timer

UneProcedureLongue

msgbox "Temps écoulé :" & Timer - StartTime

'--------------------------------------------------

Function UneProcedureLongue
Dim UneChaine, I, Cpt

'Fabrique une chaine longue
UneChaine=String(15000,"a") & String(15000,"b")

'... vraiment longue
UneChaine = UneChaine & UneChaine & UneChaine

'Test le nombre de fois que le premier caractère
'd'une chaine est trouvé.
For I = 1 To len(UneChaine)
  if mid(UneChaine,i,1)=mid(UneChaine,1,1) then Cpt=Cpt+1
Next

End Function
```

Fichier 042_fctTimer2.vbs

COMMENT OPTIMISER (Rendre plus rapide) CE PROGRAMME ? ?

Solution à mettre en î uvre:

Remplacer, dans la mesure du possible, toutes les expressions calculées dans la boucle.
En effet, lôinstruction Len est évaluée à chaque itération , et le premier caractère de la chaîne peut être calculé une fois pour toutes et non à chaque itération.

Exercice « Optimiser »
(Fichier 043_fctTimer3.vbs - solution hors support -)

Mettre en î uvre cette solution (gain dôenviron 20%, selon la machine).

LES TABLEAUX

Jusquà présent, nous avions la possibilité de stocker une valeur dans une variable.
Mais si nous voulions mémoriser, par exemple, les sept jours de la semaine, il fallait créer sept variables (Lundi, Mardi é).
Il serait bien pratique de pouvoir disposer dûne variable qui permette de stocker un **ensemble** de valeurs.
Un peu magique dites-vous ? Pas vraiment, cette fonctionnalité existe dans presque tous les langages ; on parle alors de **tableau**.
Un tableau est un ensemble d'éléments indexés de façon séquentielle et ayant le même type de données. Chaque élément d'un tableau est repéré par un numéro d'index unique. Les changements effectués sur un seul élément d'un tableau n'affectent pas les autres éléments.

Un tableau peut contenir les sept jours de la semaine , mais il faut dûabord le dimensionner :
Dim UnTableau(6)
Noter bien que le premier **indice** commence à zéro.
Maintenant, la variable UnTableau est dimensionnée pour accepter un panel de valeurs.
Par exemple : UnTableau(0)= « Lundi »
UnTableau(1)= « Mardi »
UnTableau(2)= « Mercredi »
é
UnTableau(6)= « Dimanche »

Exercice 12
Faire un programme qui demande un N° de jour entre 1 et 7 (avec contrôle de saisie effectué avec doé loop) et qui affiche le jour correspondant.

Fichier 044_tableaux1.vbs

Vous êtes vous déjà demandé comment certaines applications proposaient un truc au hasard au démarrage ?
Elles utilisent une fonction très utilisée dans les jeux :
La fonction RND

Enregistrer et tester le code suivant

```
option explicit
Dim MyValue, Response
Randomize   ' Initialise le générateur de nombres aléatoires.
Do Until Response = vbNo
   MyValue = Int((6 * Rnd) + 1)    ' Génère une valeur aléatoire
' entre 1 et 6.
   MsgBox MyValue
   Response = MsgBox ("Recommencer? ", vbYesNo)
Loop
```

Fichier 045_fctRdn1.vbs

Exercice 13
Un programme demande une date, sous la forme JOUR MOIS et renvoie le signe zodiacal correspondant.
Vous pouvez soit le faire complètement, soit « réparer » le code suivant qui a été malencontreusement attaqué par un virus é (fichier 046_tableaux2.vbs)

```
option explicit
JourSaisi=left(Saisie,instr(Saisie," "))

Zodiaque(0,0)=21  : Zodiaque(1,0)=21
```

```
Zodiaque(0,1)="mars"    : Zodiaque(1,1)="avril"
Zodiaque(0,2)=20        : Zodiaque(1,2)=21
Zodiaque(0,3)="avril"   : Zodiaque(1,3)="mai"
Zodiaque(0,4)="BELIER"  : Zodiaque(1,4)="TAUREAUX"

     if cint(JourSaisi) < cint(Zodiaque(i,0)) then
        if (i-1<0) then i=12  'Cas spécial, date inférieure au 21 Mars
      msgbox "Signe:" & Zodiaque(i-1,4)
     end if

Zodiaque(2,0)=22        : Zodiaque(3,0)=22
Zodiaque(2,1)="mai"     : Zodiaque(3,1)="juin"
Zodiaque(2,2)=21        : Zodiaque(3,2)=22
Zodiaque(2,3)="juin"    : Zodiaque(3,3)="juillet"
Zodiaque(2,4)="GEMEAUX" : Zodiaque(3,4)="CANCER"

Zodiaque(4,0)=23        : Zodiaque(5,0)=0
Zodiaque(4,1)="juillet" : Zodiaque(5,1)="--"
Zodiaque(4,2)=22        : Zodiaque(5,2)=0
Zodiaque(4,3)="aout"    : Zodiaque(5,3)="--"
Zodiaque(4,4)="LION"    : Zodiaque(5,4)="--"

Zodiaque(11,0)=20
Zodiaque(11,1)="fevrier"
Zodiaque(11,2)=20
Zodiaque(11,3)="mars"
Zodiaque(11,4)="POISSON"

dim Saisie, JourSaisi, MoisSaisi
dim iResultat, i

Zodiaque(6,0)=0         : Zodiaque(7,0)=0
Zodiaque(6,1)="--"      : Zodiaque(7,1)="--"
Zodiaque(6,2)=0         : Zodiaque(7,2)=0
Zodiaque(6,3)="--"      : Zodiaque(7,3)="--"
Zodiaque(6,4)="--"      : Zodiaque(7,4)="--"

for i=0 to 11
next

Zodiaque(8,0)=0         : Zodiaque(9,0)=0
Zodiaque(8,1)="--"      : Zodiaque(9,1)="--"
Zodiaque(8,2)=0         : Zodiaque(9,2)=0
Zodiaque(8,3)="--"      : Zodiaque(9,3)="--"
Zodiaque(8,4)="--"      : Zodiaque(9,4)="--"

Saisie = inputbox("Entrer le jour et le mois de votre naissance ( ex : 22
avril )")

  if Zodiaque(i,1)=MoisSaisi then
     if cint(JourSaisi) >= cint(Zodiaque(i,0)) then
        msgbox "Signe:" & Zodiaque(i,4)
     end if
  end if

dim Zodiaque(11,5)
MoisSaisi=right(Saisie,len(Saisie)-len(JourSaisi))
```

Solution de l'exercice N°12

```
option explicit
Dim TableauJour(6), UnChiffre

TableauJour(0)="Lundi"
TableauJour(1)="Mardi"
TableauJour(2)="Mercredi"
TableauJour(3)="Jeudi"
TableauJour(4)="Vendredi"
TableauJour(5)="Samedi"
TableauJour(6)="Dimanche"

Do

  UnChiffre=inputbox ( "Entrer un N° de jour entre 1 et 7")

loop while UnChiffre < 1 OR UnChiffre > 7

msgbox TableauJour(UnChiffre - 1 )
```

Fichier 044_tableaux1.vbs

Solution de l'exercice N°13

```
option explicit

dim Zodiaque(11,5)

Zodiaque(0,0)=21         : Zodiaque(1,0)=21
Zodiaque(0,1)="mars"     : Zodiaque(1,1)="avril"
Zodiaque(0,2)=20         : Zodiaque(1,2)=21
Zodiaque(0,3)="avril"    : Zodiaque(1,3)="mai"
Zodiaque(0,4)="BELIER"   : Zodiaque(1,4)="TAUREAUX"

Zodiaque(2,0)=22         : Zodiaque(3,0)=22
Zodiaque(2,1)="mai"      : Zodiaque(3,1)="juin"
Zodiaque(2,2)=21         : Zodiaque(3,2)=22
Zodiaque(2,3)="juin"     : Zodiaque(3,3)="juillet"
Zodiaque(2,4)="GEMEAUX"  : Zodiaque(3,4)="CANCER"

Zodiaque(4,0)=23         : Zodiaque(5,0)=0
Zodiaque(4,1)="juillet"  : Zodiaque(5,1)="--"
Zodiaque(4,2)=22         : Zodiaque(5,2)=0
Zodiaque(4,3)="aout"     : Zodiaque(5,3)="--"
Zodiaque(4,4)="LION"     : Zodiaque(5,4)="--"

Zodiaque(6,0)=0          : Zodiaque(7,0)=0
Zodiaque(6,1)="--"       : Zodiaque(7,1)="--"
Zodiaque(6,2)=0          : Zodiaque(7,2)=0
Zodiaque(6,3)="--"       : Zodiaque(7,3)="--"
Zodiaque(6,4)="--"       : Zodiaque(7,4)="--"

Zodiaque(8,0)=0          : Zodiaque(9,0)=0
Zodiaque(8,1)="--"       : Zodiaque(9,1)="--"
Zodiaque(8,2)=0          : Zodiaque(9,2)=0
Zodiaque(8,3)="--"       : Zodiaque(9,3)="--"
Zodiaque(8,4)="--"       : Zodiaque(9,4)="--"
```

```
Zodiaque(11,0)=20
Zodiaque(11,1)="fevrier"
Zodiaque(11,2)=20
Zodiaque(11,3)="mars"
Zodiaque(11,4)="POISSON"

dim Saisie, JourSaisi, MoisSaisi
dim iResultat, i

Saisie = inputbox("Entrer le jour et le mois de votre naissance ( ex : 22
avril )")
JourSaisi=left(Saisie,instr(Saisie," "))
MoisSaisi=right(Saisie,len(Saisie)-len(JourSaisi))

for i=0 to 11
  if Zodiaque(i,1)=MoisSaisi then
     if cint(JourSaisi) >= cint(Zodiaque(i,0)) then
        msgbox "Signe:" & Zodiaque(i,4)
     end if
     if cint(JourSaisi) < cint(Zodiaque(i,0)) then
        if (i-1<0) then i=12  'Cas spécial, date inférieure au 21 Mars
      msgbox "Signe:" & Zodiaque(i-1,4)
     end if
  end if
next
```

fichier 046_tableaux2.vbs

LE PASSAGE DE PARAMETRES

Lôutilisation de sous programmes est permanente.
Ils rendent le programme plus simple, plus facilement modifiable et plus robuste.
Cela entraîne nécessairement une bonne gestion des paramètres.
Il existe deux façons de passer des paramètres :
Par valeur et par référence.

Paramètres par valeur

Passer une variable par valeur signifie que la routine va créer une copie de la variable.
Ainsi, la variable originelle ne sera pas modifiée.
Cela nôempêche pas que lôon puisse modifier la copie à volonté.
On dit aussi que la variable est passée « en entrée ».

Enregistrer et tester le code suivant

```
option explicit

dim UneVar

UneVar=4

test UneVar
msgbox "UneVar=" & UneVar

sub test(byval UnParam)
 UnParam=UnParam+1
 msgbox "UnParam=" & UnParam
end sub
```

On travaille sur une **copie** de la variable

fichier 047_byval1.vbs

Toutes les fois ou il nôest pas nécessaire de modifier la variable originelle, il est **INDISPENSABLE** de passer celle-ci par **valeur**.
Pour des raisons liées à la fois à la compréhension du code et à la robustesse (car ainsi on est certain de ne pas modifier involontairement des variables ; on parle quelquefois « dôeffets de bords »).

Paramètres par référence

Passer un paramètre par référence signifie que celui-ci peut (et même **doit**) être modifié par la routine.
On dit aussi que la variable est passée « en entrée-sortie ».
Les puristes feront une distinction entre les paramètres :

- en entrée
- en sortie uniquement
- en entrée/sortie

Modifier le fichier 047_byval1.vbs en remplaçant « byval » par « byref » , enregistrer (048_byref1.vbs) et tester le programme .

Création de modules indépendants

Afin de ne pas réinventer la roue à chaque programme, il est important de pouvoir réutiliser des routines.
Le copier-coller fonctionne très bien ,mais lôinconvénient de ce procédé est que lorsque vous améliorez une routine, il faut alors modifier tous les sources dans lesquels la routine est utilisée.
Une fonctionnalité très pratique consiste à isoler les routines dans des fichiers « a part », qui seront uniques ; on parle souvent « dôincludes ».

Créer un sous-répertoire supplémentaire nommé « includes ».

Enregistrer le code suivant

```
Include "testmessage.vbs"

Main


Sub Main
  dim x
  x="Hello" :  message1 x
  msgbox x
  x="Hello" :  message2 x
  msgbox x

End Sub

Sub Include(file)
  Dim fso, f
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set f = fso.OpenTextFile(file, 1)
  str = f.ReadAll
  f.Close
  ExecuteGlobal str
End Sub
```

Fichier \includes\01_include.vbs

Enregistrer le code suivant

```
option explicit

sub message1(byval UneChaine)
  msgbox "Depuis un autre fichier ( byval )" & vbcrlf & UneChaine
end sub

sub message2(byref UneChaine)
  msgbox "Depuis un autre fichier ( byref )" & vbcrlf & UneChaine
  UneChaine=UneChaine & "!!!"  'Modification du paramètre
end sub
```

Fichier \includes\testmessage.vbs

Puis tester « 01_include.vbs »

TP « Tri »

Lôutilisation des tris est fréquente, et on trouve aujourdôhui des composants qui permettent dôeffectuer ces opérations.
Mais il nôest pas toujours possibles de les utiliser ; cela dépends du type de langage utilisé, des contraintes fixées par lôenvironnement du projet, des performances attendues etc. .

En tous les cas, cela reste un excellent exercice pédagogique.

A vous dôinventer votre routine de tri !

(1) Créer une routine de tri dôun tableau dôentiers.
Appeler cette routine depuis un autre source via la technique des includes.

A moins que vous ne soyez un expert en tri ou un génie méconnu, votre routine souffrira, en performance, de la comparaison avec les routines mises au point depuis des années et que lôon peut trouver sur le net.

A vous de les trouver puisé

(2) Créer un module qui contiendra deux routines de tri différentes (par exemple , un tri a bulles et un tri dichotomique).

Utiliser ces routines, via des includes, dans un fichier qui traitera un tableau de 300 entiers remplis au hasard, puis affichera le temps mis par chacune des 2 routines.
(fichier 02_include.vbs et tris.vbs - solution hors support -)

LES CARRES MAGIQUES

11	10	4	23	17
18	12	6	5	24
25	19	13	7	1
2	21	20	14	8
9	3	22	16	15

Observez attentivement ce carré de 5 X 5 cases.
La somme des lignes fait exactement 65.
Mais la somme des colonnes fait également 65.
Plus surprenant encore, la somme des diagonales principales fait aussi 65 !

Ce carré est dôautant plus magique quôaucun nombre nôest présent plus dôune fois, et que leur liste est composée des nombres entiers compris entre 1 et 25é .

Essayez de composer un carré magique de 3 X 3

Facile ? !

Essayez maintenant avec un carré de 7 X 7 é .

Les précédents carrés avaient un nombre impair de cases.
Ils étaient dôordre 5, 3 et 7.
Pour les carrés dôordre impair, il existe une méthode de résolution automatique, et il serait vraiment magique dôutiliser un programme pour les faire automatiquement non ? !

Analysons dôabord comment les construire manuellement.

Constatons dôabord que pour un ordre n, le nombre maximum appelé NbMax est égal à n au carré.
Donc, pour un carré de 7, NbMax=49.

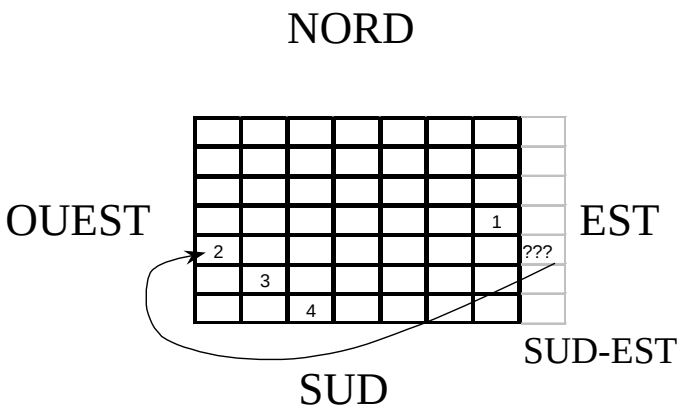
La somme des lignes ou des colonnnes ou des diagonales correspond à ((NbMax+1)/2) X n.
Pour un carré de 7, le total sera 25 X 7 soit 175.

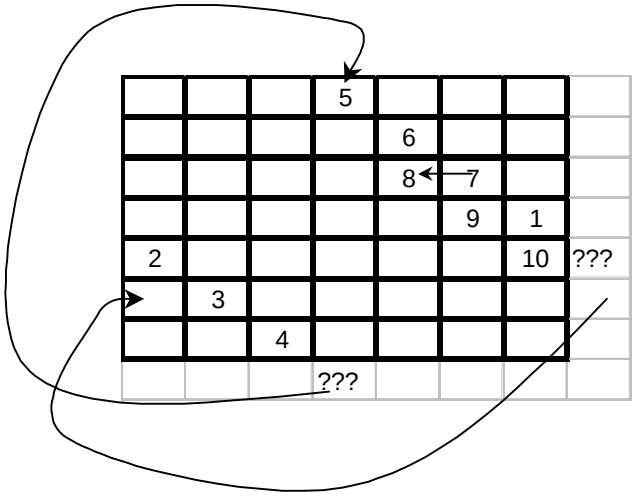
Commencer par placer le chiffre 1 au milieu de la dernière colonne.

PRINCIPE.

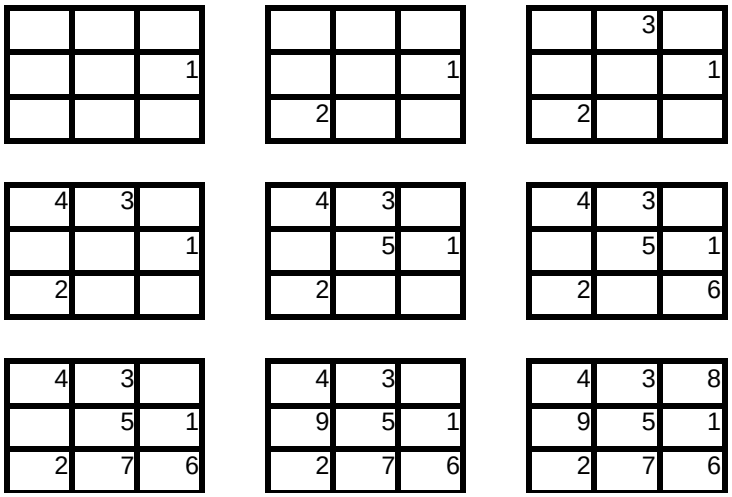
A partir du chiffre 1, tester la case située en Sud-Est

a) Si la case est vide, y placer le chiffre suivant.
b) Si la case est pleine, se placer une case à lôouest (à gauche donc).
c) Si on se trouve « dans le vide » en EST, se placer sur la première colonne, une rangée plus bas.
d) Si on se trouve « dans le vide » en SUD, se placer sur la première rangée, une colonne plus loin.
e) Si on se trouve « dans le vide » en SUD-EST (ça nôarrive quôune fois) , se placer une case à lôouest (à gauche donc).
RECOMMENCER LôOPERATION A PARTIR DU DERNIER NOMBRE PLACE.





Voici toutes les étapes pour un carré magique dôordre 3



Continuer la construction du carré magique dôordre 7 avant de vérifier la solution page suivante

							175
22	21	13	5	46	38	30	175
31	23	15	14	6	47	39	175
40	32	24	16	8	7	48	175
49	41	33	25	17	9	1	175
2	43	42	34	26	18	10	175
11	3	44	36	35	27	19	175
20	12	4	45	37	29	28	175
175	175	175	175	175	175	175	175

Il est important de comprendre la mécanique de cette construction avant de commencer le programme.

Entraînez-vous sur le papier à composer un carré dôordre 5.

Exercice 14

A vous de créer un programme qui fabrique des carrés dôordre impair (lôordre est demandé au départ).

Solution de l'exercice N°14

```
option explicit
'-------------------------------------------------------
'--- Déclaration des variables
dim Ordre, Carre, NbMax, Pivot, Totaux

'-------------------------------------------------------
'--- Initialisation des variables
Ordre=Cint(inputbox ("Entrer l'ordre du carré inpair"))
ReDim Carre(Ordre, Ordre)

NbMax=Ordre * Ordre
Pivot=(NbMax+1)/2
Totaux=Pivot * Ordre

'-------------------------------------------------------
'--- Affichage d'informations ----------------------
dim msg
msg= "Calcul d'un carré magique de " & Ordre & " par " & ordre & vbcrlf
msg=msg & "Donc allant de 1 à " & NbMax & vbcrlf
msg=msg & "Chiffre pivot : " & Pivot & vbcrlf
msg=msg & "Totaux attendus ( lig/col/diag. ) : " & Totaux & vbcrlf

msg=msg & vbcrlf & "VOULEZ-VOUS CONTINUER ?" & vbcrlf

if msgbox( msg, vbyesno, "INFOS" )=vbno then Wscript.quit(0)
'-------------------------------------------------------

RemplisDeZeros

Construction

Affichage

"""""""""""""""
'FIN DU PROGRAMME"
"""""""""""""""
'==============
'PROCEDURES   =
'==============

'============================================================
sub Affichage
        dim i,j

        'JUSTE POUR TESTER --DEBUT--
        'Carre(1, 1) = "*"
        'Carre(1, 2) = "8"
        'Carre(1, 3) = "9"
        'Carre(2, 1) = "21"
        'JUSTE POUR TESTER --FIN  --

        msg=""

        for i=0 to Ordre ï  1
                for j=0 to Ordre ï  1
                        msg=msg & Carre(i,j) & "  "
                next
                msg=msg & vbcrlf
        next
```

```
	msgbox msg
end sub
'=========================================================
Sub RemplisDeZeros

	dim i,j
	for i=0 to Ordre ï 1
		for j=0 to Ordre ï 1
			Carre(i,j)=0 'JUSTE POUR TEST
		Next
	Next
End sub
'=========================================================
Sub Construction
	'Positionnement du premier chiffre
	dim Ran,Col
	dim NextRan,NextCol
	dim i

	Ran=int(Ordre/2)
	Col=Ordre-1
	Carre(Ran,Col) = "1"

	for i=2 to NbMax
		NextRan=Ran+1 :NextCol=Col+1

		if NextRan=Ordre and NextCol=Ordre then
			'On se trouve sorti en SUD-EST
			NextRan=Ran
			NextCol=Col-1
		elseif NextCol=Ordre then
			'On se trouve sorti en EST
			NextRan=Ran+1
			NextCol=0
		elseif NextRan=Ordre  then
			'On se trouve sorti en SUD
			NextRan=0
			NextCol=Col+1
		end if

	    'Si la case SUD-EST est déjà affectée ...
	'...déplacement d'une colonne à gauche par
	     'rapport au dernier chiffre placé
	If Carre(NextRan, NextCol) <> 0 Then
	      NextCol = NextCol ï 2
	   NextRan = Ran
	     End If

		Ran=NextRan : Col=NextCol
		Carre(Ran,Col) = i

		'Enlever le commentaire '-> pour voir la construction
		'au fur et à mesure !!
		'-> Affichage
	next
end sub
'=========================================================
```


The
End.

ANNEXE

Qu'appelle-t'on langage informatique?

On appelle langage informatique un langage destiné à décrire l'ensemble des actions consécutives qu'un ordinateur doit exécuter.

Les langages naturels (l'anglais, le français) représentent l'ensemble des façons qu'ont un groupe d'individu de communiquer. Les langages servant aux ordinateurs à communiquer n'ont rien à voir avec des langages informatiques, on parle dans ce cas de *protocoles*, ce sont deux notions totalement différentes. Un langage informatique est une façon pratique pour nous (humains) de donner des instructions à un ordinateur.

Un langage informatique est rigoureux: à CHAQUE instruction correspond UNE action du processeur. Le langage utilisé par le processeur, c'est-à-dire les données telles qu'elles lui arrivent, est appelé langage machine. Il s'agit d'une suite de 0 et de 1 (du binaire) mais pour plus de "clarté il peut être décrit en hexadécimal. Toutefois le langage machine n'est pas compréhensible facilement par l'humain moyen é
Ainsi il est plus pratique de trouver un langage intermédiaire, compréhensible par l'homme, qui sera ensuite transformé en langage machine pour être exploitable par le processeur.

L'assembleur est le premier langage informatique qui ait été utilisé. Celui-ci est encore très proche du langage machine mais il permet déjà d'être plus compréhensible. Toutefois un tel langage est tellement proche du langage machine qu'il dépend étroitement du type de processeur utilisé (chaque type de processeur peut avoir son propre langage machine). Ainsi un programme développé pour une machine ne pourra pas être *porté* sur un autre type de machine (on désigne par « *portable* » un programme qui peut être utilisé sur un grand nombre de machine). Pour pouvoir l'utiliser sur une autre machine il faudra alors parfois réécrire entièrement le programme!

Un langage informatique a donc plusieurs avantages:

- il est plus facilement compréhensible que le langage machine
- il permet une plus grande portabilité, c'est-à-dire une plus grande facilité d'adaptation sur des machines de types différents

Les langages informatiques peuvent grossièrement se classer en deux catégories: les langages interprétés et les langages compilés.

Langage interprété

Un langage informatique est par définition différent du langage machine. Il faut donc le traduire pour le rendre intelligible du point de vue du processeur. Un programme écrit dans un langage interprété a besoin d'un programme auxiliaire (l'interpréteur) pour traduire au fur et à mesure les instructions du programme.

Langage compilé

Un programme écrit dans un langage dit "compilé" va être traduit une fois pour toutes par un programme annexe (le compilateur) afin de générer un nouveau fichier qui sera autonome, c'est-à-dire qui n'aura plus besoin d'un programme autre que lui pour s'exécuter (on dit d'ailleurs que ce fichier est exécutable).
Un programme écrit dans un langage compilé a comme avantage de ne plus avoir besoin, une fois compilé, de programme annexe pour s'exécuter. De plus, la traduction étant faite une fois pour toute, il est plus rapide à l'exécution.
Toutefois il est moins souple que programme écrit avec un langage interprété car à chaque modification du fichier source (fichier intelligible par l'homme: celui qui va être compilé) il faudra recompiler le programme pour que les modifications prennent effet.
D'autre part, un programme compilé a pour avantage de garantir la sécurité du code source. En effet, un langage interprété, étant directement intelligible (lisible), permet à n'importe qui de connaître les secrets de fabrication d'un programme et donc de copier le code voire de le modifier. Il y a donc risque de non-respect des droits d'auteur. D'autre part, certaines applications sécurisées nécessitent la confidentialité du code pour éviter le piratage (transaction bancaire, paiement en ligne, communications sécurisées, ...).

Langages intermédiaires

Certains langages appartiennent en quelque sorte aux deux catégories (LISP, Java, Python, ..) car le programme écrit avec ces langages peut dans certaines conditions subir une phase de compilation intermédiaire vers un fichier écrit dans un langage qui n'est pas intelligible (donc différent du fichier source) et non exécutable (nécessité d'un interpréteur). Les applets Java, petits programmes insérés parfois dans les pages Web, sont des fichiers qui sont compilés mais que l'on ne peut exécuter qu'à partir d'un navigateur internet (ce sont des fichiers dont l'extension est .class).

Quelques exemples de langages couramment utilisés
Voici une liste non exhaustive de langages informatiques existants.

Langage	**Domaine de prédilection**	**Compilé/interprété**
ADA	Temps réel (armement, transport)	langage compilé
Basic (VBScript)	é é	langage interprété
C	é é	langage compilé
C++	Programmation objet	langage compilé
Cobol	Gestion	langage compilé
Fortran	Calcul	langage compilé
Java	Programmation orientée internet	langage intermédiaire
MATLAB	Calcul mathématique	langage interprété
Mathematica	Calcul mathématique	langage interprété
LISP	Intelligence artificielle	langage intermédiaire
Pascal	Enseignement	langage compilé
Prolog	Intelligence artificielle	langage interprété
PHP	Web (serveur)	langage intermédiaire
Perl	Traitement de chaînes de caractères	langage interprété

Présentation du binaire

Vers la fin des années 30, Claude Shannon démontra qu'à l'aide de "contacteurs" (interrupteurs) fermés pour "vrai" et ouverts pour "faux" on pouvait effectuer des opérations logiques en associant le nombre " 1 " pour "vrai" et "0" pour "faux".
Ce langage est nommé langage binaire. C'est avec ce langage que fonctionnent les ordinateurs. Il permet d'utiliser deux chiffres (0 et 1) pour faire des nombres. L'homme travaille quant à lui avec 10 chiffres (0,1,2,3,4,5,6,7,8,9), on parle alors de base décimale.

Le bit

Bit signifie "binary digit", c'est-à-dire 0 ou 1 en numérotation binaire. C'est la plus petite unité d'information manipulable par une machine.
On peut les représenter physiquement:

- par une impulsion électrique, qui, lorsqu'elle atteint une certaine valeur, correspond à la valeur 1.
- par des trous dans une surface
- grâce à des bistables, c'est-à-dire des composants qui ont deux états d'équilibre (un correspond à l'état 1, l'autre à 0)

Avec un bit on peut avoir soit 1, soit 0.
Avec 2 bits on peut avoir quatre états différents (2*2):

0 0
0 1
1 0
1 1

Avec 3 bits on peut avoir huit états différents (2*2*2):

0 0 0
0 0 1
0 1 0
0 1 1
1 0 0
1 0 1
1 1 0
1 1 1

Avec huit bits on a 2*2*2*2*2*2*2*2=256 possibilités, c'est ce que l'on appelle un octet.

2^7 =128	2^6 =64	2^5 =32	2^4 =16	2^3 =8	2^2 =4	2^1 =2	2^0 =1
0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1

Le plus petit nombre est 0, le plus grand est 255, il y a donc 256 possibilités
Cette notion peut être étendue à n bits, on a alors 2^n possibilités.

L'octet

L'octet est une unité d'information composée de 8 bits. Il permet de stocker un caractère, telle qu'une lettre, un chiffre ...
Ce regroupement de nombres par série de 8 permet une lisibilité plus grande, au même titre que l'on apprécie, en base décimale, de regrouper les nombres par trois pour pouvoir distinguer les milliers. Par exemple le nombre 1 256 245 est plus lisible que 1256245.
Une unité d'information composée de 16 bits est généralement appelée *mot* (en anglais *word*)
Une unité d'information de 32 bits de longueur est appelée *double mot* (en anglais *double word*, d'où l'appelation *dword*).

KiloOctets, MégaOctets

Un kilo-octet (Ko) ne vaut pas 1000 octets mais 2^{10}=1024 octets
Un méga-octet (Mo) vaut 2^{10} Ko=1024 Ko=1 048 576 octets

Les opérations en binaire

Les opérations arithmétiques simples telles que l'addition, la soustraction et la multiplication sont faciles à effectuer en binaire.

L'addition en binaire

L'addition en binaire se fait avec les mêmes règles qu'en décimale:
On commence à additionner les bits de poids faibles (les bits de droite) puis on a des retenues lorsque la somme de deux bits de mêmes poids dépasse la valeur de l'unité la plus grande (dans le cas du binaire: 1), cette retenue est reportée sur le bit de poids plus fort suivant...
Par exemple:

```
  0 1 1 0 1
+ 0 1 1 1 0
- - - - - -
  1 1 0 1 1
```

La multiplication en binaire

La multiplication se fait entre bits de même poids, avec le même système de retenue qu'en décimale. La table de multiplication en binaire est très simple:

- 0x0=0
- 0x1=0
- 1x0=0
- 1x1=1

Par exemple:

```
  0 0 1 0 1
x 0 0 0 1 0
- - - - - -
  0 1 0 1 0
```


Dans *Métaphysique*, **Aristote** écrit : *" C'est en tout cas une vérité évidente que l'expression* être ou ne pas être *présente une signification définie, de sorte que rien ne saurait* être ainsi et non ainsi. "

Les types de données de VBScript

VBScript comprend un seul type de données nommé **Variant**. Un **Variant** est un type de données spécial qui peut contenir différent types d'information, en fonction de son utilisation. Parce que le **Variant** est le seul type de données de VBScript, c'est aussi le type de données retourné par toutes les fonctions de VBScript.

À la base, un **Variant** peut contenir soit des informations numériques soit des informations de type chaîne de caractères. Un **Variant** se comporte comme un nombre lorsqu'il est utilisé dans un contexte numérique et comme une chaîne lorsqu'il est utilisé dans un contexte de chaîne. Si vous travaillez avec des données qui ressemblent à des nombres, VBScript suppose qu'il s'agit de nombres et agit de façon appropriée. De même, si vous travaillez avec des données qui ne peuvent être que des chaînes, VBScript les traitent comme des chaînes. Bien sûr, vous pouvez toujours traiter les nombres comme des chaînes en les encadrant avec des guillemets (" ").

Sous-types Variant

Au-delà de la simple distinction nombre/chaîne, un **Variant** peut distinguer différents types d'information numérique. Par exemple, certaines informations numériques représentent une date ou une heure. Lorsque ces informations sont utilisées avec d'autres données de date ou d'heure, le résultat est toujours exprimé sous la forme d'une date ou d'une heure. Bien sûr, vous disposez aussi d'autres types d'information numérique, des valeurs booléennes jusqu'aux grands nombres à virgule flottante. Ces différentes catégories d'information qui peuvent être contenues dans un **Variant** sont des sous-types. Dans la plupart des cas, vous placez simplement vos données dans un **Variant** et celui-ci se comporte de la façon la plus appropriée en fonction de ces données.

Le tableau suivant présente les différents sous-types susceptibles d'être contenus dans un **Variant**.

Sous-type	Description
Empty	Le **Variant** n'est pas initialisé. Sa valeur est égale à zéro pour les variables numériques et à une chaîne de longueur nulle ("") pour les variables chaîne.
Null	Le **Variant** contient intentionnellement des données incorrectes.
Boolean	Contient **True** (vrai) ou **False** (faux).
Byte	Contient un entier de 0 à 255.
Integer	Contient un entier de -32 768 à 32 767.
Currency	-922 337 203 685 477,5808 à 922 337 203 685 477,5807.
Long	Contient un entier de -2 147 483 648 à 2 147 483 647.
Single	Contient un nombre à virgule flottante en précision simple de -3,402823E38 à -1,401298E-45 pour les valeurs négatives ; de 1,401298E-45 à 3,402823E38 pour les valeurs positives.
Double	Contient un nombre à virgule flottante en précision double de -1,79769313486232E308 à -4,94065645841247E-324 pour les valeurs négatives ; de 4,94065645841247E-324 à 1,79769313486232E308 pour les valeurs positives.
Date (Time)	Contient un nombre qui représente une date entre le 1er Janvier 100 et le 31 Décembre 9999.
String	Contient une chaîne de longueur variable limitée à environ 2 milliards de caractères.
Object	Contient un objet.
Error	Contient un numéro d'erreur.

Vous pouvez utiliser des fonctions de conversion pour convertir les données d'un sous-type vers un autre. En outre, la fonction **VarType** retourne des informations sur la façon dont vos données sont stockées dans un **Variant**.

Examinons comment les données de divers types sont placés dans la mémoire.
Commençons par les caractères. Tout d'abord, si on veut placer des caractères dans la mémoire, il faut les coder en binaire, c'est-à-dire associer à chaque caractère une série de bits qui le représente. Mais de combien de bits a-t-on besoin pour chaque caractère ? Eh bien cela dépend du nombre de caractères différents il y a à représenter. Comptons: il y a 26 lettres de l'alphabet majuscules, 26 minuscules, 10 chiffres, et une vingtaine de caractères de ponctuation et de signes mathématiques. On arrive à un total de 92 symboles différents. Avec 2 bits, on peut coder 4 symboles différents (00, 01, 10 et 11). Avec n bits, on peut coder 2^n symboles différents. Pour coder 92 symboles, on a donc besoin de 7 bits, car $2^7 = 128$ ($2^6 = 64$, donc 6 bits sont insuffisants).
Cependant, avec le développement de l'informatique, on eu tôt fait d'avoir besoin de toutes sortes de caractères supplémentaires pour les accents dans diverses langues ainsi que pour le dessin de cadres à l'écran. On a donc pris un code sur 8 bits, qui permet de coder 256 symboles différents. C'est d'ailleurs une des raisons qui ont fait que la taille des cellules mémoires s'est standardisée sur 8 bits par cellule. Et maintenant, vous vous demandez peut-être comment fait-on pour coder les divers caractères. On utilise le code ASCII.

Regardons maintenant les nombres entiers. La différence avec les caractères est notable: le nombre de caractères est limité (moins de 256), alors que le nombre d'entiers est illimité! Et nous voyons là les limitations de l'ordinateur: nous ne pouvons pas représenter tous les nombres entiers, mais seulement un nombre fini. La quantité de nombres entiers qui peuvent être représentés dépend donc du nombre de bits que nous pouvons lui attribuer. Avec 1 octet, nous pouvons représenter 256 nombres (de 0 à 255 ou de -128 à +127). Avec 2 octets, nous pouvons représenter $2^{16} = 65536$ nombres (de 0 à 65535 ou de -32768 à +32767). Avec 4 octets, nous pouvons représenter $2^{32} = 4294967296$ nombres (de 0 à environ 4 milliards ou d'environ - 2 milliards à environ + 2 milliards). Cependant, chaque nombre entier est représenté de façon exacte. Lors de l'écriture de programmes, ces limites doivent être prises en compte.
Regardons enfin les nombres réels. La situation est, dans une certaine mesure, similaire à celle des nombres entiers car on doit représenter une infinité de nombres avec un nombre fini de bits. Cependant, à la différence des nombres entiers, la représentation des nombres n'est pas exacte mais seulement une approximation car, entre deux nombres réels, il existe une infinité de nombres réels. Cependant, plus le nombre de bits alloués à la représentation des nombres réels est grand, plus la précision de la représentation est bonne.

Jeu de caractères (0 - 127)

Code	Car	Code	Car	Code	Car	Code	Car
0		32	[espace]	64	@	96	`
1		33	!	65	A	97	a
2		34	"	66	B	98	b
3		35	#	67	C	99	c
4		36	$	68	D	100	d
5		37	%	69	E	101	e
6		38	&	70	F	102	f
7		39	'	71	G	103	g
8	**	40	(	72	H	104	h
9	**	41	)	73	I	105	i
10	**	42	*	74	J	106	j
11		43	+	75	K	107	k
12		44	,	76	L	108	l
13	**	45	-	77	M	109	m
14		46	.	78	N	110	n
15	_	47	/	79	O	111	o
16	_	48	0	80	P	112	p
17	_	49	1	81	Q	113	q
18	_	50	2	82	R	114	r
19		51	3	83	S	115	s
20		52	4	84	T	116	t
21		53	5	85	U	117	u
22	_	54	6	86	V	118	v
23	_	55	7	87	W	119	w
24	_	56	8	88	X	120	x
25	_	57	9	89	Y	121	y
26	_	58	:	90	Z	122	z
27		59	;	91	[	123	{
28	_	60	<	92	\	124	|
29		61	=	93	]	125	}
30	-	62	>	94	^	126	~
31		63	?	95	_	127	•

** Les valeurs 8, 9, 10 et 13 sont converties respectivement en espace arrière, tabulation, saut de ligne et retour chariot. Ces caractères ne sont pas représentés graphiquement, mais en fonction de l'application utilisée, ont un impact sur l'affichage du texte.

Remerciements :
Pour certains extraits :
Jean-François Pillou (« http://www.commentcamarche.net/ » licence GNU FDL).
Michel Couprie et Denis Bureau ("Initiation à la programmation structurée" écrit pour le Groupe ESIEE.)