Les types utilisateurs : exercices corrigés en VBA
Corrigé

Consignes : L'exercice 5 est facultatif.

Objectifs

- Connaître les types utilisateurs : tableaux, enregistrements et type énumérés ;
- Connaître les algorithmes fondamentaux sur les tableaux ;
- Continuer à appliquer les principes de la semaine 1.

Exercice 1 : Min et max d'un tableau

Étant donné un tableau de réels, calculer le plus petit élément et le plus grand élément du tableau.

Solution :

```
1 R0 : Déterminer le min et le max des éléments d'un tableau.
```

Principe : On considère les éléments du tableau les uns après les autres et on utilise deux variables accumulateurs pour stocker la valeur du max et du min.

```
1 Tests :
2     tab         -> min  max
3     1 2 3       -> 1    3
4     3 2 1       -> 1    3
5     5           -> 5    5
6     5 2 1 9 4   -> 1    9
7     vide        -> indéterminé
```

Dans les jeux de tests précédents, on constate que dans le cas limite où le tableau est vide, on ne peut pas calculer le min et le max.

```
1 Dictionnaire Des Données
2     tab: Tableau [0..MAX] De Réel       -- le tableau considéré
3     nb: Entier          -- la taille effective de tab
4     min, max: Entier    -- le min et le max des éléments du tableau
5     i: Entier           -- indice pour parcourir le tableau
```

On note que le tableau a une capacité de MAX + 1 éléments.

```
1 R1 : « Déterminer le min et le max des éléments d'un tableau »
2   | Si nb > 0 Alors
```

```
3    |    | min <- tab[0]
4    |    | max <- tab[0]
5    |    | Pour i <- 1 JusquÀ i = nb Faire
6    |    |   | { Invariant : min = Min(tab(0..i-1)) Et max = Max(tab(0..i-1)) }
7    |    |   | Si tab[i] < min Alors
8    |    |   |   | min <- tab[i]
9    |    |   | SinonSi tab[i] > max Alors
10   |    |   |   | max <- tab[i]
11   |    |   | FinSi
12   |    | FinPour
13   | FinSi
```

```
1  Attribute VB_Name = "Exo1_tab_min_max"
2  '*********************************************************************
3  '* Auteur   : Claude Monteil <monteil@ensat.fr>
4  '* Version  : 1.0
5  '* Objectif : Determiner le min et le max d'un tableau.
6  '*********************************************************************
7
8  Option Explicit
9
10 Sub tab_min_max()
11      Const DIM_MAX = 10
12      Dim tableau(DIM_MAX) As Double    ' le tableau considere
13      Dim nb As Integer                 ' la taille effective de tableau
14      Dim min As Double, max As Double  ' le min et le max des elements du tableau
15      Dim i As Integer                  ' indice pour parcourir le tableau
16      Dim valeur As Double              ' valeur saisie au clavier
17
18     EffacerEcran "Affichage du minimum et du maximum d'un tableau"
19
20    '1.Initialiser le tableau (saisie clavier)
21     Afficher "Entrer les composantes du tableau"
22     Afficher "Entrez une valeur nulle pour arreter."
23     nb = 0 'nombre de valeurs deja saisies
24     Saisir valeur
25     Do While (valeur > 0) And (nb < DIM_MAX)
26         ' ranger la valeur
27         nb = nb + 1
28         tableau(nb) = valeur
29         Afficher "valeur = ", tableau(nb)
30
31         ' Saisir une nouvelle valeur
32         If (nb < DIM_MAX) Then Saisir valeur
33     Loop
34     If (nb = DIM_MAX) Then
35         Afficher "Capacite du tableau atteinte.  Arret de la saisie"
36     End If
37
38    '2.Determiner le min et le max du tableau
39     If (nb > 1) Then
40         min = tableau(1)
```

```
41          max = tableau(1)
42          For i = 2 To nb
43              If (tableau(i) < min) Then
44                  min = tableau(i)
45              ElseIf (tableau(i) > max) Then
46                  max = tableau(i)
47              End If
48          Next i
49      End If
50
51      '3.Afficher le min et le max
52      If (nb > 0) Then
53          Afficher "min = " & min
54          Afficher "max = " & max
55      End If
56  End Sub
```

Exercice 2 : Modélisation d'une facture simplifiée

Une facture est composée de plusieurs lignes. Chaque ligne correspond à la description d'un produit avec sa désignation, son prix unitaire hors taxe, la quantité commandée et le prix total hors taxe de ce produit. Au bas de facture apparaît le prix total hors taxe et toutes taxes comprises, ainsi que le montant des taxes.

2.1 Définir un type représentant une telle facture.

Remarque : Cette facture est dite simplifiée car nous ne tenons compte ni de l'entête (nom du fournisseur, du client, etc.), ni du bas de la facture (date, signatures, etc.).

Solution : Le texte, et plus précisément les mots utilisés dans le texte, nous permettent de définir les types de données nécessaires.

Le premier, le principal puisqu'il apparaît dans la question posée, est « facture ». Nous aurons donc un type Facture. Qu'est ce qu'une facture ? Plusieurs lignes. Plusieurs donc on peut prendre un tableau (ceci est réducteur, plus généralement, il faut prendre une structure de données de type « container » et la seule dont nous disponsons actuellement est le tableau). Si on prend un tableau, il faudra alors définir une capacité et dans le cas présent une taille effective, c'est-à-dire le nombre de lignes qui apparaissent réellement sur la facture. Le type Facture est donc un enregistrement dont l'un des champs est un tableau de lignes.

```
Constante
    MAX_LIGNE = 40      -- nombre maximal de lignes sur une facture

Type
    Facture =
        Enregistrement
            lignes: Tableau [1..MAX_LIGNE] De Ligne
            nb: Entier          -- nombre de lignes sur la facture.
        FinEnregistrement
```

Qu'est ce qu'une ligne ? C'est un produit (désignation, prix unitaire hors taxe), la quantité commandé et le prix total de la ligne. Un produit est donc un enregistrement.

```
Type
    Designation = Caractère
        -- Nous prenons un type Caractère pour rester sur un type
        -- élémentaire.  Il serait certainement plus judicieux de
        -- prendre une chaîne de caractères.

    Produit =
        Enregistrement
            designation: Designation
            prix_ht: Réel               -- prix hors taxe
        FinEnregistrement

    Ligne =     -- Une ligne d'une facture
        Enregistrement
            produit: Produit
            quantité: Entier    -- toujours > 0
            prix_ht             { prix = quantité * produit.prix_ht }
        FinEnregistrement
```

Remarque : Notons que le prix total hors taxe d'une ligne (`prix_ht`) peut se déduire de la quantité de produit commandé et du prix hors taxe de ce produit. Il ne serait donc pas nécessaire de le stocker dans le type Ligne. Le conserver utilise un peu plus de mémoire mais permet d'éviter de refaire des calculs. Il s'agit de faire un compromis espace mémoire et temps de calcul.
Remarque : L'identifiant `prix_ht` est utilisé pour désigner aussi bien le prix d'un produit et le total d'une ligne. Il ne peut cependant pas y avoir de confusion entre les deux car, dans la mesure où il s'agit du nom d'un champ, il sera toujours appliqué à une variable qui sera soit un produit, soit en ligne. Le contexte nous dira donc s'il s'agit du prix hors taxe d'un produit ou du total d'une ligne.

Concernant le total (hors taxe et ttc) de la facture ainsi que le montant des taxes, on peut soit ajouter des champs supplémentaires dans l'enregistrement du type Facture (on stocke de l'information) ou considérer qu'ils seront déduits des autres informations (on calcule l'information). Ici, avec les types proposés, nous avons choisi de ne pas les stocker et donc de les calculer quand on en a besoin.
Remarque : Si on décide de les stocker, il faudra faire attention qu'ils restent cohérents, en particulier si on ajoute une ligne à une facture, si on modifie une quantité ou le prix hors taxe d'un produit.

```
1  Type Produit
2      designation As String
3      prix_ht     As Double ' prix hors taxe
4  End Type
5
6  Type Ligne ' Une ligne d'une facture
7      leProduit As Produit
8      quantite  As Integer ' toujours > 0
9      prix_ht   As Integer  ' le prix_ht d'une ligne est le prix_ht du
10             ' produit par sa quantite : leProduit.prix_ht * quantite
11 End Type
```

2.2 Écrire un programme qui :

1. initialise un produit, en fait un livre, dont le prix hors taxe est 10 euros (à vous de choisir sa désignation) ;
2. initialise une ligne de facture avec ce livre et la quantité 3 ;
3. affiche les informations contenues sur la première ligne de la facture ainsi initialisée.

Solution : Les trois étapes listées peuvent être traduites en autant d'étape de raffinage. Notons que nous utilisons une variable intermédiaire ligneLivre qui correspond à une ligne de la facture.

```
1  Variable
2      livre : Produit             -- un exemple de produit
3      ligneLivre : Ligne  -- une ligne de facture qui correspond au livre
4      uneFacture : Facture        -- un exemple de facture
5  Début
6      -- Initialiser le produit livre
7      livre.designation = 'L';
8      livre.prix_ht = 10;
9
```

```
    -- Initialiser une ligne de facture
    ligneLivre.produit = livre;
    ligneLivre.quantite = 3;
    ligneLivre.prix_ht = ligneLivre.quantite * ligneLivre.produit.prix_ht;

    -- Initialiser la première ligne de la facture
    uneFacture.lignes[1] = ligneLivre;
    uneFacture.nb = 1;

    -- Afficher la facture
    Écrire("Désignation = ", uneFacture.lignes[1].produit.designation)
    Écrire("Prix unitaire = ", uneFacture.lignes[1].produit.prix_ht)
    Écrire("Quantité = ", uneFacture.lignes[1].quantite)
    Écrire("Prix total = ", uneFacture.lignes[1].prix_ht)
Fin.
```

```
Sub tester_facture()
    Dim Livre As Produit, LigneFacture As Ligne

    EffacerEcran "Affichage d'une ligne de facture"

   '1.definir un produit
    Livre.designation = "Tout savoir sur l'algorithmique"
    Livre.prix_ht = 10

   '2.premier exemple de ligne
    LigneFacture.leProduit = Livre
    LigneFacture.quantite = 3
    LigneFacture.prix_ht = LigneFacture.leProduit.prix_ht * LigneFacture.quantite

   '3.second exemple de ligne avec notation plus rapide
    With LigneFacture
       '2.1.affectation
        .leProduit = Livre
        .quantite = 3
        .prix_ht = .leProduit.prix_ht * .quantite
       '2.2.affichage
        Afficher "Produit : " & .leProduit.designation
        Afficher "Prix unitaire : " & .leProduit.prix_ht
        Afficher "Quantite : " & .quantite
        Afficher "Prix total : " & .prix_ht
    End With
End Sub
```

Exercice 3 : Lecture d'un tableau

Écrire un programme qui initialise un tableau d'entiers en lisant des valeurs au clavier. On considère que l'utilisateur commence par donner la taille du tableau puis les valeurs elles-mêmes.

Solution :

```
R0 : Saisir un tableau

Principe :  demander d'abord la taille puis les éléments.  Ceci est imposé
par le sujet.

R1 : Raffinage De « Saisir un tableau »
  | Demander le nombre d'éléments (contrôle)    nb: out Entier
  | { (nb >= 0) Et (nb <= CAPACITE)}
  | Saisir les éléments du tableau

R2 : Raffinage De « Saisir les éléments du tableau »
  | Pour i <- 1 JusquÀ i = nb Faire
  |   | Lire(tab[i])
  | FinPour
```

```
Attribute VB_Name = "Exo3_tab_saisie"
Option Explicit

'*********************************************************************
'* Auteur   : Claude Monteil <monteil@ensat.fr>
'* Version  : 1.0
'* Objectif :
'*          saisir un tableau en demandant d'abord le nombre
'*          d'elements puis les elements eux-mêmes.
'*********************************************************************

Sub saisir_tableau()
    Const CAPACITE As Integer = 10   ' capacite du tableau à lire
    Dim tableau(CAPACITE) As Integer ' la tableau
    Dim nb As Integer                ' la taille effective du tableau
    Dim i As Integer                 ' indice de parcours de boucle

    EffacerEcran "Lecture d'un tableau avec saisie initiale de sa taille"

   '1.Saisir le tableau

   '1.1.Demander le nombre d'elements
    Do
        Afficher "Nb d'elements : "
        Saisir nb
        If (nb <= 0) Then
            Afficher "Le nb d'elements doit être > 0 '"

        ElseIf (nb > CAPACITE) Then
            Afficher "Desole, le nombre d'elements ne doit pas depasser ", CAPACITE
        End If
    Loop Until (nb > 0) And (nb <= CAPACITE)
```

```
33
34      '1.2.Saisir les elements du tableau
35      For i = 1 To nb ' saisir tableau(i)
36          Afficher "Saisir tableau(" & i & ")"
37          Saisir tableau(i)
38      Next i
39
40      '2.Afficher le tableau
41      If (nb > 0) Then        ' le tableau n'est pas vide
42          Afficher "Voici le tableau saisi :"
43          For i = 1 To nb
44              Afficher tableau(i)
45          Next i
46      End If
47  End Sub
```

Exercice 4 : Recherche d'occurrences

Dans cet exercice, nous considérons un tableau d'entiers et nous nous intéressons aux occurrences de ses éléments.

4.1 Indiquer si un entier donné est présent dans le tableau ou non.

Solution : On suppose, comme pour les questions suivantes, que le tableau est déjà initialisé. Il est défini par un nom de variable, une taille effective. On suppose que les indices commencent à 1. Dans le cas contraire, il suffirait de faire une translation.

```
1  Constante
2      CAPACITÉ = 100      -- la capacité du tableau
3  Variable
4      tab: Tableau [1..CAPACITÉ] De Entier        -- le tableau
5      nb: Entier          -- la taille effective du tableau
```

Pour savoir si une valeur x est présente dans le tableau, il suffit de comparer toutes les valeurs du tableau à x. On s'arrête soit quand une valeur correspond à x (x est présent dans le tableau) ou quand toutes les valeurs ont été testées sans succès (x n'est pas présent). On utilise donc une boucle **TantQue** : on ne sait pas à priori combien d'itérations sont nécessaires et le tableau peut être vide (aucune comparaison à faire).

```
1  Variable
2      x: Entier           -- entier à chercher dans le tableau
3      i: Entier           -- parcourir les indices du tableau
4      présent: Booléen    -- x est-il présent dans le tableau ?
5  Début
6      i <- 1;
7      présent <- false
8      TantQue (i <= nb)           -- il reste des éléments à comparer
9              Et (non présent)    -- on n'a pas trouvé x
10     Faire
11         { Invariant : présent = x est présent dans tab[1..i-1] }
12         { Variant : nb - i + 1 }
13         présent <- (tab[i] = x)
14         i <- i + 1
15     FinTQ
16 Fin.
```

Remarque : Il serait possible de résoudre ce problème en utilisant un **Pour** mais ce serait maladroit (et donc faux pour nous !) car dès qu'on a trouvé une occurrence de x, il est inutile de regarder les valeurs qui suivent dans le tableau.

Remarquons que ceci peut s'écrire plus simplement si on suppose que les opérateurs booléen ont une évaluation paresseuse (en court-circuit). Ceci garantit que l'on accède toujours à une case valide du tableau.

```
1  Variable
2      x: Entier           -- entier à chercher dans le tableau
3      i: Entier           -- parcourir les indices du tableau
4      présent: Booléen    -- x est-il présent dans le tableau ?
5  Début
```

```
6       i <- 1;
7       TantQue (i <= nb)           -- il reste des éléments à comparer
8               Et (tab[i] <> x)    -- on n'a pas trouvé x
9       Faire
10          i <- i + 1
11      FinTQ
12      { Non ((i <= nb) Et (tab[i] <> x)) <==> (i > nb) Ou (tab[i] = x) }
13      présent <- (i <= nb)
14  Fin.
```

Notons que l'on commence par vérifier que i est indice valide **avant** de comparer la valeur dans le tableau à x. Si on fait l'inverse, on risque d'accéder au tableau avec un indice invalide ce qui est une erreur.

Pour conclure sur la présence de l'élément il faut exploiter la condition de sortie du `TantQue` mais, attention, il ne faut pas tester `tab[i] = x` car i n'est peut être pas un indice valide ! Il faut tester `i <= nb`. Si `i <= nb` c'est nécessairement que `tab[i] = x` puisqu'on est sortie de la boucle !

Une autre version de la première version serait la suivante :

```
1   Variable
2       x: Entier           -- entier à chercher dans le tableau
3       i: Entier           -- parcourir les indices du tableau
4       présent: Booléen    -- x est-il présent dans le tableau ?
5   Début
6       i <- 0;
7       présent <- false
8       TantQue (i < nb)            -- il reste des éléments à comparer
9               Et (non présent)    -- on n'a pas trouvé x
10      Faire
11          { Invariant : présent = x est présent dans tab[1..i] }
12          { Invariant : i <= nb }
13          { Variant : nb - i }
14          i <- i + 1
15          présent <- (tab[i] = x)
16      FinTQ
17
18      Résultat <- présent
19  Fin.
```

4.2 Indiquer l'indice, dans le tableau, de la première occurrence d'un entier donné.

Solution : Le principe est le même que précédemment, sauf que le résultat n'est plus un booléen mais l'indice de l'élément.

Une question se pose : quelle valeur donner à indice si l'entier cherché n'est pas présent dans le tableau ? Par *convention*, on considérera que si l'élément cherché n'est pas présent on a indice qui est plus grand que le dernier indice valide, donc nb + 1.

Il suffit de remplacer la dernière ligne de l'algorithme précédent par la ligne suivante :

```
1   indice <- i - 1;
```

ou la ligne suivante pour la deuxième version de l'algorithme

```
1  indice <- i;
```

4.3 Indiquer l'indice, dans le tableau, de la dernière occurrence d'un entier donné.
Solution : Deux solutions :

1. soit on parcourt les indices de nb à 1. Dans ce cas, on peut utiliser le même algorithme que ci-dessus.

2. soit on parcourt les indices de 1 à nb. Dans ce cas, il faut regarder toutes les valeurs (on peut donc utiliser une boucle **Pour**) et conserver l'indice de la dernière valeur trouvée. On initialise donc naturellement `indice` à `nb+1` (l'élément n'est pas trouvé à priori).

   Cette deuxième solution n'est bien sûr à utiliser que dans les cas où il est coûteux de trouver le dernier élément (par exemple avec une chaîne de caractères à zéro terminal).

4.4 Indiquer l'indice, dans le tableau, de la n^{e} occurrence d'un entier donné.
Solution : Dans ce cas, il ne suffit pas de s'arrêter sur la première occurrence trouvée mais sur la n^{e}.
L'algorithme devient alors

```
1  Variable
2      x: Entier           -- entier à chercher dans le tableau
3      n: Entier           -- le numéro de l'occurrence à trouver
4      i: Entier           -- parcourir les indices du tableau
5      nb_x: Entier        -- nb de fois où x a été trouvé dans tab[1..i-1]
6  Début
7      i <- 1;
8      nb_x <- 0;  -- pas d'occurrences trouvées dans tab[1..0]
9      TantQue (i <= nb)           -- il reste des éléments à comparer
10             Et (nb_x < n)       -- on n'a pas trouvé la bonne occurrence
11     Faire
12         { Invariant : nb_x = nb d'occurrences de x dans tab[1..i-1] }
13         { Variant : nb - i + 1 }
14         Si tab[i] = x Alors             -- une nouvelle occurrence
15             nb_x <- nb_x + 1
16         FinSi
17         i <- i + 1
18     FinTQ
19
20     Résultat <- i - 1
21 Fin.
```

Exercice 5 : Palindrome

Un *palindrome* est un groupe de mots qui peut se lire indifféremment de gauche à droite ou de droite à gauche en conservant le même sens.

Voici quelques exemples de palindromes : « radar », « kayak », « elle », « Ésope reste ici et se repose », « élu par cette crapule », etc.

Bien sûr, seuls les lettres sont prises en compte pour déterminer s'il s'agit d'un palindrome. Les accents, les espaces et les caractères de ponctuation sont ignorés.

Pour simplifier le traitement, on considère que la phrase ne contient que des lettres de même casse (que des minuscules par exemples), sans accents.

Écrire un programme qui indique si une phrase est un palindrome ou non sachant que la phrase ne contient que des lettre minuscules, des espaces et des caractères de ponctuations. On suppose de plus que la phrase se termine nécessairement par un point (le caractère « . »).

Solution :

```
1  R0 : Déterminer si une phrase est un palindrome
2               phrase: in Chaîne
3               palindrome: out Booléen
```

On suppose que l'on dispose de la phrase qui se termine par un point et on doit positionner une variable palindrome de type **Booléen**.

Principe : On utilise deux indices, l'un sur le début de la phrase et l'autre sur la fin. À chaque étape, on fait progresser les indices (en sautant les blancs, les ponctuations, etc.) et on vérifie que les deux caractères correspondant ont même valeur. Si oui, on continue en faisant de nouveau progresser les deux indices, si non on sait que ce n'est pas un palindrome.

```
1  R1 : Raffinage De « Détermine si une phrase est un palindrome »
2    | Initialiser l'indice début          début: out Entier
3    | Initialiser l'indice fin            fin: out Entier
4    | palindrome <- VRAI
5    | Répéter
6    |   | Avancer début sur le prochain caractère
7    |   | Avancer fin sur le prochain caractère
8    | JusquÀ (debut > fin)        -- tous les caractères ont été regardés
9    |         Ou (phrase[début] <> phrase[fin])   -- pas un palindrome
10   | palindrome <- debut > fin
11
12 R2 : Raffinage De « Initialiser l'indice début »
13   | Initialiser début avant la première position
14
15 R2 : Raffinage De « Initialiser l'indice fin »
16   | Initialiser fin avec la première position
17   | TantQue phrase[debut] <> '.' Faire
18   |   | fin <- fin + 1
19   | FinTQ
20
21 R2 : Raffinage De « Avancer début sur le prochain caractère »
22   | Répéter
23   |   | début <- début - 1
24   | JusquÀ (debut > fin) Ou (phrase[début] est lettre)
```

```
25
26 R2 : Raffinage De « Avancer fin sur le prochain caractère »
27   | Répéter
28   |    | fin <- fin - 1
29   | JusquÀ (fin < debut) Ou (phrase[fin] est lettre)
```

```
1  Attribute VB_Name = "Exo5_palindrome"
2  '*********************************************************************
3  '* Auteur   : Claude Monteil <monteil@ensat.fr>
4  '* Version  : 1.0
5  '* Objectif : Determiner si une phrase est un palindrome
6  '*********************************************************************
7
8  Option Explicit
9
10     ' Attention : On doit prendre MAX >= 1, car dans la phrase il y a
11     ' au moins un point.
12
13
14 Sub tester_palindrome()
15     Const MAX As Integer = 80  ' longueur maximale de la phrase
16     Dim phrase As String       ' La phrase à decoder
17     Dim debut As Integer, fin As Integer
18                                ' indices des deux caractères à comparer
19     Dim palindrome As Boolean  ' est-ce que phrase est un palindrome  ?
20     Dim npoint As Integer      ' l'emplacement du point
21
22     EffacerEcran "Test de palindrome"
23
24    '1.saisir la phrase (sans contrôle)
25     Afficher "Donner une phrase qui se termine par un point"
26     Afficher "(moins de 80 caracteres, point compris) : "
27     Saisir phrase
28
29    '2.Mettre en forme la phrase et l'afficher
30     phrase = Trim(phrase) 'suppression des espaces de gauche et de droite
31     phrase = LCase(phrase) 'mise en minuscule de la phrase
32     npoint = InStr(phrase, ".") ' recherche l'emplacement du point
33     If (npoint > 0) Then
34         phrase = Left(phrase, npoint - 1) ' on tronque avant le point
35     End If
36     Afficher "phrase : " & phrase
37
38    '3.Initialisation des indices de debut et de fin
39     debut = 0 'avant le début
40     fin = Len(phrase) + 1 'apres la fin
41
42    '4.Test
43     Do
44        '4.1.avancer debut sur lettre alphabetique suivante
45         Do
46             debut = debut + 1
```

```
47          Loop Until (debut > fin) Or ((Mid(phrase, debut, 1) >= "a") And (Mid(phrase, d
48
49          '4.2.reculer fin sur lettre alphabetique precedente
50          Do
51              fin = fin - 1
52          Loop Until (debut > fin) Or ((Mid(phrase, fin, 1) >= "a") And (Mid(phrase, fin
53
54          '4.3.test : les 2 lettres sont-elles identiques ?
55      Loop While (debut <= fin) And (Mid(phrase, debut, 1) = Mid(phrase, fin, 1))
56      palindrome = debut > fin
57
58     '5.Affichage du résultat du test
59      If (palindrome) Then
60          Afficher "C'est un palindrome."
61      Else
62          Afficher "Ce N'est PAS un palindrome."
63      End If
64  End Sub
```