Microsoft® Visual Basic® Scripting Edition

Variables de VBScript

Qu'est-ce qu'une variable ?

Une variable est un symbole pratique qui fait référence à un emplacement en mémoire où vous stockez des informations du programme pouvant varier au cours de l'exécution du script. Par exemple, vous pouvez créer une variable que vous nommez `NbClics` pour stocker le nombre de fois qu'un utilisateur clique sur un objet d'une page Web particulière. L'endroit où se trouve effectivement la variable dans la mémoire de l'ordinateur est sans importance. Ce qui compte, c'est le nom de la variable grâce auquel vous pouvez lire ou modifier sa valeur. Dans VBScript, les variables appartiennent toujours à un type de données fondamental : Variant.

Déclaration des variables

Vous déclarez des variables explicitement dans votre script par l'intermédiaire des instructions Dim, Public et Private. Par exemple :

```
Dim DegrésFahrenheit
```

Vous pouvez déclarer plusieurs variables en séparant leur nom par une virgule. Par exemple :

```
Dim Haut, Bas, Gauche, Droite
```

Vous pouvez aussi déclarer une variable implicitement en utilisant simplement son nom dans votre script. Toutefois, cette technique n'est pas vraiment recommandée car une seule faute de syntaxe dans le nom de la variable peut donner des résultats incorrects à l'exécution. Pour cette raison, utilisez l'instruction Option Explicit pour rendre obligatoire la déclaration explicite des variables. L'instruction Option Explicit doit être la première de votre script.

Restrictions de notation

Les noms de variable suivent les règles de dénomination standard de VBScript. Un nom de variable :

- doit commencer par un caractère alphabétique.
- ne peut pas contenir de point.
- ne doit pas excéder 255 caractères.
- doit être unique à l'intérieur de la même portée.

Portée et durée de vie des variables

La portée d'une variable dépend de l'endroit où elle est déclarée. Lorsque vous déclarez une variable dans une procédure, seul le code de cette procédure peut lire ou modifier la valeur de cette variable. Elle a une portée locale et elle est désignée comme variable de niveau procédure. Si vous déclarez une variable en dehors d'une procédure, elle peut être reconnue par toutes les procédures de votre script. C'est alors une variable de niveau script, et sa portée est l'ensemble du script.

La période d'existence d'une variable est sa durée de vie. La durée de vie d'une

variable de niveau script s'étend de la déclaration de la variable à la fin de l'exécution du script. Au niveau procédure, une variable existe tant que la procédure est en cours. À la fin de la procédure, la variable est détruite. Les variables locales sont idéales pour les stockages temporaires de la procédure. Les mêmes noms de variables locales sont utilisables dans des procédures différentes parce que chacun n'est visible que dans sa procédure de déclaration.

Affectation de valeurs aux variables

Vous affectez une valeur à une variable en créant une expression de la forme suivante : la variable figure du côté gauche de l'expression et la valeur à affecter se trouve du côté droit. Par exemple :

```
B = 200
```

Variables scalaires et variables tableau

La plupart du temps, vous voulez simplement affecter une valeur à une variable que vous avez déclarée. Une variable contenant une valeur unique est une variable scalaire. Dans d'autres cas, il est pratique d'affecter plusieurs valeurs liées à une variable unique. Vous créez alors une variable contenant une série de valeurs. Dans ce cas, il s'agit d'une variable tableau. Les variables tableau et les variables scalaires sont déclarées de la même façon, sauf que la déclaration d'une variable tableau fait suivre le nom de la variable de parenthèses (). Dans l'exemple ci-dessous, on déclare un tableau à une dimension contenant 11 éléments :

```
Dim A(10)
```

Bien que le nombre entre parenthèses indique 10, ce tableau contient en fait 11 éléments car, dans VBScript, tous les tableaux commencent à zéro. Dans un tableau à base zéro, le nombre d'éléments correspond toujours au nombre entre parenthèses plus un. La taille de ce tableau est fixe.

Vous affectez les données à chaque élément en utilisant un index dans le tableau. En commençant à zéro et en finissant à 10, l'affectation des données aux éléments se présente comme suit :

```
A(0) = 256
A(1) = 324
A(2) = 100
. . .
A(10) = 55
```

De la même façon, vous lisez les données d'un élément particulier du tableau grâce à son index. Par exemple :

```
. . .
UneVariable = A(8)
. . .
```

Les tableaux ne sont pas limités à une seule dimension. Vous pouvez avoir jusqu'à 60 dimensions bien que la plupart des utilisateurs ne puissent comprendre un tableau de plus de trois ou quatre dimensions. Les différentes dimensions sont déclarées en séparant à l'intérieur des parenthèses des nombres représentant leur taille. Dans l'exemple ci-dessous, la variable `MaTable` est un tableau à deux dimensions constitué de 6 lignes et 11 colonnes:

```
Dim MaTable(5, 10)
```

Dans les tableaux à deux dimensions, le premier nombre est le nombre de lignes, le second est le nombre de colonnes.

Vous pouvez aussi déclarer un tableau dont la taille change au cours de l'exécution du script. Il s'agit alors d'un tableau dynamique. Ce tableau est initialement déclaré au sein d'une procédure en utilisant l'instruction Dim ou l'instruction ReDim. Toutefois, pour un tableau dynamique, la taille et le nombre de dimensions ne figurent pas entre parenthèses. Par exemple :

```
Dim MonTableau()
ReDim UnAutreTableau()
```

Pour utiliser un tableau dynamique, vous devez employer ReDim pour déterminer le nombre de dimensions et la taille de chaque dimension. Dans l'exemple ci-dessous, ReDim définit la taille initiale du tableau dynamique à 25. Une instruction ReDim suivante redimensionne le tableau à 30 mais utilise le mot clé Preserve pour préserver le contenu du tableau pendant l'opération.

```
ReDim MonTableau(25)
. . .
ReDim Preserve MonTableau(30)
```

Le nombre de redimensionnements d'un tableau n'est pas limité mais lorsque vous réduisez sa taille, vous perdez les données correspondant aux éléments supprimés.

Microsoft® Visual Basic® Scripting Edition

Didacticiel VBScript

Référence du langage VBScript

Microsoft® Visual Basic® Scripting Edition

Référence du langage VBScript

Didacticiel VBScript
Version Information

Caractéristiques	de VBScript
Liste des mots clés	Abs, fonction
Constantes	Couleur, constantes
Fonctions	Abs, fonction
Méthodes	Add, méthode
Objets	Dictionary, objet
Opérateurs	Addition (+), opérateur
Propriétés	Attributes, propriété
Instructions	Call, instruction

Microsoft® Visual Basic® Scripting Edition

Référence du langage

Informations sur la version

Le tableau suivant indique la version de Microsoft Visual Basic Scripting Edition utilisée par les applications hôtes.

Hôte	VBScript Version 1.0	VBScript Version 2.0	VBScript Version 3.0
Microsoft Internet Explorer 3.0	**x**		
Microsoft Internet Information Server 3.0		**x**	
Microsoft Internet Explorer 4.0			**x**
Microsoft Internet Information Server 4.0			**x**
Microsoft Windows Scripting Host 1.0			**x**
Microsoft Outlook 98			**x**

Informations légales

Documentation Microsoft

Les informations contenues dans ce document pourront faire l'objet de modifications sans préavis. Sauf mention contraire, les sociétés, les noms et les données utilisés dans les exemples sont fictifs. L'utilisateur est tenu d'observer la réglementation relative aux droits d'auteur applicable dans son pays. Aucune partie de ce manuel ne peut être reproduite ou transmise à quelque fin ou par quelque moyen que ce soit, électronique ou mécanique, sans la permission expresse et écrite de Microsoft Corporation. Si toutefois, votre seul moyen d'accès est électronique, le présent document vous autorise une et une seule copie.

Microsoft peut détenir des brevets, avoir déposé des demandes d'enregistrement de brevets ou être titulaire de marques, droits d'auteur ou autres droits de propriété intellectuelle portant sur tout ou partie des éléments qui font l'objet du présent document. Sauf stipulation expresse contraire d'un contrat de licence écrit de Microsoft, la fourniture de ce document n'a pas pour effet de vous concéder une licence sur ces brevets, marques, droits d'auteur ou autres droits de propriété intellectuelle.

Microsoft, MS-DOS, MS, Windows, Windows NT, ActiveX, JScript, Microsoft Press, Visual Basic, Visual C++, Visual J++, Win32 et Win32s sont soit des marques de Microsoft Corporation, soit des marques déposées de Microsoft Corporation, aux États-Unis d'Amérique et/ou dans d'autres pays.

Java est une marque de Sun Microsystems, Inc.

Alpha AXP est une marque de Digital Equipment Corporation.

Pentium est une marque déposée d'Intel Corporation.

Macintosh et QuickTime sont des marques déposées d'Apple Computer, Inc.

Les autres noms de produits et de sociétés mentionnés sont des marques de leurs propriétaires respectifs.

Microsoft® Visual Basic® Scripting Edition

Qu'est-ce que VBScript ?

Didacticiel VBScript
Suivant

Microsoft Visual Basic Scripting Edition, le dernier membre de la famille de langages de programmation Visual Basic, permet d'utiliser des scripts actifs dans de nombreux environnements, comme les scripts clients dans Microsoft Internet Explorer et les scripts de serveur Web dans Microsoft Internet Information Server.

Facile à utiliser et à apprendre

Si vous connaissez déjà Visual Basic ou Visual Basic pour Applications, l'utilisation de VBScript sera un jeu d'enfant. Si vous ne connaissez pas Visual Basic, l'apprentissage des concepts de VBScript constitue un premier pas vers la programmation avec l'ensemble des langages de la famille Visual Basic. Bien que ces quelques pages Web puissent vous apprendre VBScript, elles ne vous enseigneront pas la programmation. Pour aborder la programmation, reportez-vous à *Step by Step*, publié par Microsoft Press.

Les scripts ActiveX actifs

VBScript parle aux applications hôtes qui utilisent ActiveX™ Scripting. Avec ActiveX Scripting, les navigateurs et les autres applications hôtes ne nécessitent pas de code d'intégration spécial pour chaque composant de script. ActiveX Scripting permet à un hôte de compiler les scripts, d'obtenir et d'appeler des points d'entrée et de gérer l'espace de nom disponible au développeur. Avec ActiveX Scripting, les fournisseurs de langages peuvent créer des exécutables de langage standard pour les scripts. Microsoft assurera le support des exécutables pour VBScript. Microsoft collabore avec différents acteurs d'Internet pour définir le standard ActiveX Scripting afin que les moteurs de script soient interchangeables. ActiveX Scripting est utilisé dans Microsoft® Internet Explorer et dans Microsoft® Internet Information Server.

VBScript dans d'autres applications et navigateurs

En tant que développeur, vous pouvez bénéficier d'une licence gratuite d'implémentation de la source VBScript dans vos produits. Microsoft fournit les implémentations binaires de VBScript pour l'API Windows® 32 bits, l'API Windows 16 bits et Macintosh®. VBScript est intégré aux navigateurs Web. VBScript et ActiveX Scripting sont également utilisables comme langage de script dans d'autres applications.

Ajout de code VBScript dans une page HTML

Vous pouvez utiliser l'élément SCRIPT pour ajouter du code VBScript dans une page HTML.

Balise <SCRIPT>

Le code VBScript est encadré par des balises <SCRIPT>. Voici un exemple de procédure qui teste une date de livraison :

```
<SCRIPT LANGUAGE="VBScript">
<!--
   Function LivraisonPossible(Dt)
      LivraisonPossible = (CDate(Dt) - Now()) > 2
   End Function
-->
</SCRIPT>
```

Le code est encadré par des balises <SCRIPT> de début et de fin. L'attribut LANGUAGE indique le langage de script. Vous devez spécifier le langage car les navigateurs peuvent utiliser d'autres langages de script. Remarquez que la fonction Livraison est encadrée par des balises de commentaire (<!-- et -->). Ainsi, les navigateurs qui ne comprennent par la balise <SCRIPT> n'affichent pas le code.

Commel'exemple représente une fonction générale—elle n'est pas liée à un contrôle particulier d'une feuille—vous pouvez la placer dans la section HEAD de la page :

```
<HTML>
<HEAD>
<TITLE>Passez votre commande</TITLE>
<SCRIPT LANGUAGE="VBScript">
<!--
   Function LivraisonPossible(Dt)
      LivraisonPossible = (CDate(Dt) - Now()) > 2
   End Function
-->
</SCRIPT>
</HEAD>
<BODY>
...
```

Vous pouvez utiliser des blocs SCRIPT n'importe où dans une page HTML, par exemple, dans la section BODY ou la section HEAD. Toutefois, nous vous conseillons de placer tout le code d'usage commun dans la section HEAD. Ainsi, vous êtes sûr qu'il sera lu et décodé avant d'être requis par les appels de la section BODY.

Exception à cette règle : lorsque vous voulez fournir du code de script en ligne au sein des feuilles pour répondre aux événements des objets de la feuille. Par exemple, vous pouvez incorporer du code de script pour répondre à un clic de bouton dans une feuille :

```
<HTML>
<HEAD>
<TITLE>Tester les événements de bouton</TITLE>
</HEAD>
<BODY>
<FORM NAME="Feuille1">
   <INPUT TYPE="Button" NAME="Bouton1" VALUE="Clic">
   <SCRIPT FOR="Bouton1" EVENT="onClick" LANGUAGE="VBScript">
      MsgBox "Bouton cliqué!"
   </SCRIPT>
</FORM>
</BODY>
</HTML>
```

La plus grande partie de votre code apparaîtra dans des procédures Sub ou Function et sera appelée uniquement quand le code que vous avez écrit causera son exécution. Toutefois, vous pouvez placer du code en dehors des procédures mais toujours dans un bloc SCRIPT. Ce code s'exécute une seule fois, au chargement de la page HTML. Ceci vous permet d'initialiser des données ou de changer dynamiquement la présentation de votre page Web lors de son chargement.

Microsoft® Visual Basic® Scripting Edition

Types de données de VBScript

Quel sont 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.

Microsoft® Visual Basic® Scripting Edition

Référence du langage

True

Voir aussi

Description

Le mot clé True a une valeur égale à -1.

Microsoft® Visual Basic® Scripting Edition

Référence du langage

False

Voir aussi

Description

Le mot clé False a une valeur égale à 0.

Conversion, fonctions

Asc, fonction

Voir aussi

Description

Retourne le code de caractère ANSI correspondant à la première lettre d'une chaîne.

Syntaxe

Asc(*string*)

L'argument *string* correspond à toute expression de chaîne valide. Si *string* ne contient aucun caractère, une erreur d'exécution se produit.

Note

Remarque Une autre fonction (**AscB**) peut être utilisée avec les données de type octet contenues dans une chaîne. Au lieu de retourner le code de caractère du premier caractère, **AscB** retourne le premier octet. **AscW** est disponible sur les plates-formes 32 bits qui utilisent des caractères Unicode. Elle retourne le code de caractère Unicode (large), ce qui évite une conversion de Unicode à ANSI.

Asc , fonction

Voir aussi

Chr, fonction

Microsoft® Visual Basic® Scripting Edition

Référence du langage

Chr, fonction

Voir aussi

Description

Retourne le caractère associé au code de caractère ANSI spécifié.

Syntaxe

Chr(*charcode*)

L'argument *charcode* représente un nombre qui identifie un caractère.

Note

Les nombres de 0 à 31 sont identiques aux codes ASCII standard, non imprimables. Par exemple, Chr(10) retourne un caractère de retour à la ligne.

Remarque Une autre fonction (**ChrB**) peut être utilisée avec les données d'octet contenues dans une chaîne. Au lieu de retourner un caractère, qui peut être un ou deux octets, la fonction **ChrB** retourne toujours un seul octet. **ChrW** est disponible pour les plates-formes 32 bits utilisant des caractères Unicode. Son argument est un code de caractère Unicode (large), ce qui évite une conversion de ANSI à Unicode.

Chr, fonction

Voir aussi

Asc, fonction

Microsoft® Visual Basic® Scripting Edition

Référence du langage

Glossaire VBScript

argument

Constante, variable ou expression transmise à une procédure.

bibliothèque de types

Fichier ou composant d'un autre fichier contenant des descriptions standard d'objets, méthodes et propriétés exposés.

classe

Définition formelle d'un objet. La classe joue le rôle d'un modèle à partir duquel une instance d'un objet est créée à l'exécution. La classe définit les propriétés de l'objet et les méthodes utilisées pour contrôler le comportement de celui-ci.

code de caractère

Nombre représentant un caractère particulier d'un jeu, le jeu de caractères ASCII par exemple.

collection

Objet contenant un ensemble d'objets en relation. La position d'un objet dans la collection peut être modifiée chaque fois qu'un changement se produit dans la collection. La position dans la collection de tout objet spécifique peut donc varier.

commentaire

Texte ajouté au code par un développeur pour expliquer le fonctionnement du code. Dans Visual Basic Script, une ligne de commentaire commence habituellement par une apostrophe (') ; vous pouvez aussi utiliser le mot clé Rem suivi d'un espace.

comparaison binaire

Comparaison bit par bit des bits dont la position est identique dans deux expressions numériques.

comparaison de chaîne

Combinaison de deux séquences de caractères. Sauf spécification contraire dans la fonction de comparaison, toutes les comparaisons de chaîne sont binaires. En Anglais, les comparaisons binaires respectent la casse, au contraire des comparaisons de texte.

constante

Élément nommé qui conserve une valeur constante pendant toute l'exécution d'un programme. Vous pouvez utiliser des constantes n'importe où dans votre code à la place de valeurs réelles. Une constante peut être une chaîne ou un littéral numérique, une autre constante ou toute combinaison incluant des opérateurs arithmétiques ou logiques à l'exception de Is et de l'élévation à une puissance. Par exemple :

```
Const A = "MaChaîne"
```

---

constante intrinsèque

Constante fournie par une application. Etant donné qu'une constante intrinsèque ne peut être désactivée, vous ne pouvez pas créer de constante utilisateur portant le même nom.

---

contrôle ActiveX

Objet placé sur une feuille pour permettre ou améliorer l'interaction de l'utilisateur avec l'application. Les contrôles ActiveX comprennent des événements et peuvent être incorporés à d'autres contrôles. Leur nom de fichier comporte l'extension .ocx.

Empty

Valeur indiquant qu'aucune valeur n'a encore été affectée à une variable. Les variables Empty correspondent à 0 dans un contexte numérique ou de longueur nulle dans un contexte de chaîne.

---

erreur d'exécution

Erreur qui se produit au cours de l'exécution du code. Une erreur d'exécution se produit lorsqu'une instruction tente une opération incorrecte.

---

exécution

Phase pendant laquelle le code s'exécute. Pendant l'exécution, vous ne pouvez pas modifier le code.

---

expression

Combinaison de mots clés, d'opérateurs, de variables et de constantes qui produit une chaîne, un nombre ou un objet. Une expression peut effectuer un calcul, manipuler des caractères ou tester des données.

---

expression booléenne

Expression dont le résultat est True ou False.

expression de chaîne

Toute expression qui peut être évaluée en séquence de caractères contigus. Parmi les éléments d'une expression de chaîne peuvent figurer une fonction retournant une chaîne, une constante de chaîne, un littéral chaîne ou une variable chaîne.

---

expression de date

Toute expression pouvant être interprétée comme une date. Ceci comprend toute combinaison de littéraux de date, de nombres ou de chaînes ressemblant à des dates, et de dates retournées par des fonctions. Une expression de date est limitée aux nombres ou chaînes, sous toute forme de combinaison, qui représentent une date comprise entre le 1er janvier 100 et le 31 décembre 9999.

Les dates sont enregistrées comme partie d'un nombre réel. Les valeurs à gauche de la virgule représentent la date, tandis que les valeurs à droite représentent le temps. Les valeurs négatives correspondent aux dates antérieures au 30 décembre 1899.

---

expression numérique

Toute expression pouvant être résolue sous forme de nombre. Les éléments de l'expression peuvent inclure toute combinaison de mots clés, de variables, de constantes et d'opérateurs dont le résultat est un nombre.

---

jeu de caractères ASCII

Jeu de caractères ASCII (American Standard Code for Information Interchange) codés sur 7 bits, largement utilisé pour représenter les lettres et les symboles d'un clavier américain standard. Le jeu de caractères ASCII est identique aux 128 premiers caractères (0–127) du jeu de caractères ANSI.

---

littéral de date

Toute séquence de caractères de format valide entourée de dièses (signe #). Les formats valides incluent le format de date spécifié par les paramètres locaux pour votre code ou le format de date universel. Par exemple, #12/31/99# est le littéral de date qui correspond au 31 décembre 1999 si Anglais (États-Unis) est le paramètre local pour votre application.

Dans VBScript, le seul format reconnu est US-ENGLISH, quelle que soit la localisation réelle de l'utilisateur. Autrement dit, le format interprété est mm/jj/aaaa.

---

module de classe

Module contenant la définition d'une classe (la définition de ses propriétés et de ses méthodes).

---

mot clé

Mot ou symbole reconnu comme faisant partie du langage VBScript ; par exemple une instruction, un nom de fonction ou un opérateur.

---

niveau procédure

Décrit les instructions situées à l'intérieur d'une procédure Function ou Sub. En principe, les déclarations sont d'abord répertoriées, puis suivies des affectations et de tout autre code exécutable. Par exemple :

```
Sub MaSub() ' Cette instruction déclare une procédure Sub.
    Dim A ' Cette instruction débute le corps de la procédure.
    A = "Ma variable" ' Code de niveau procédure.
    Debug.Print A ' Code de niveau procédure.
End Sub ' Cette instruction finit la procédure Sub.
```

Notez que le code de niveau module réside à l'extérieur de tout bloc de procédure.

---

niveau script

Tout code figurant hors d'une procédure est désigné comme code de niveau script.

Nothing

Valeur spéciale indiquant qu'une variable objet n'est plus associée à un objet réel.

Null

Valeur indiquant qu'une variable ne contient aucune donnée valide. Null est le résultat :

- d'une affectation explicite de la valeur Null à une variable ;
- de toute opération entre expressions contenant la valeur Null.

---

numéro de l'erreur

Nombre entier compris entre 0 et 65 535 inclus qui correspond à la propriété Number de l'objet Err. Lorsqu'il est combiné à la propriété Name de l'objet Err, ce numéro représente un message d'erreur particulier.

---

objet ActiveX

Objet exposé aux autres applications ou outils de programmation par l'intermédiaire des interfaces Automation.

---

objet Automation

Objet exposé aux autres applications ou outils de programmation par l'intermédiaire des interfaces Automation.

opérateur de comparaison

Caractère ou symbole indiquant une relation entre deux ou plusieurs valeurs ou expressions. Ces opérateurs comprennent les signes inférieur à (<), inférieur ou égal à (<=), supérieur à (>), supérieur ou égal à (>=), différent de (<>), et égal à (=).

Is est aussi un opérateur de comparaison, mais il est exclusivement utilisé pour déterminer si une référence d'un objet est identique à une autre.

par référence

Moyen de transmettre l'adresse, au lieu de la valeur, d'un argument à une procédure. Ceci permet à la procédure d'accéder à la variable elle-même. En conséquence, la valeur de la variable peut être modifiée par la procédure à laquelle elle est transmise.

par valeur

Moyen de transmettre la valeur, au lieu de l'adresse, d'un argument à une procédure. Ceci permet à la procédure d'accéder à une copie de la variable. En conséquence, la valeur de la variable ne peut pas être modifiée par la procédure à laquelle elle est transmise.

paramètres régionaux

Informations relatives à une langue et à un pays donné qui ont une incidence particulière sur le code et le système.

- Dans le code, les paramètres régionaux affectent la langue des termes tels que les mots clés et définissent les séparateurs décimaux et les séparateurs de liste, les formats de date et l'ordre de tri des caractères spécifiques au pays.
- Les paramètres régionaux définis pour le système affectent le comportement de la fonctionnalité gérant les caractéristiques locales comme l'affichage des nombres ou la conversion des chaînes en dates. Pour les configurer, utilisez les utilitaires du Panneau de configuration fournis par le système d'exploitation.

pi

Pi est une constante mathématique approximativement égale à 3,1415926535897932.

plages de données

Chaque sous-type Variant possède une plage spécifique de valeurs autorisées :

Sous-type	**Plage**
Byte	0 à 255.
Boolean	**True** ou **False**.
Integer	-32 768 à 32 767.

Long	-2 147 483 648 à 2 147 483 647.
Single	-3,402823E38 à -1,401298E-45 pour les valeurs négatives ; 1,401298E-45 à 3,402823E38 pour les valeurs positives.
Double	-1,79769313486232E308 à -4,94065645841247E-324 pour les valeurs négatives ; 4,94065645841247E-324 à 1,79769313486232E308 pour les valeurs positives.
Currency	-922 337 203 685 477,5808 à 922 337 203 685 477,5807.
Date	1er janvier 100 au 31 décembre 9999, inclus.
Object	Toute référence **Object**.
String	Les chaînes de longueur variable peuvent contenir entre 0 et 2 milliards de caractères environ.

---

portée

Définit la visibilité d'une variable, d'une procédure ou d'un objet. Par exemple, une variable déclarée comme Public est visible pour toutes les procédures dans tous les modules. Les variables déclarées dans les procédures ne sont visibles qu'à l'intérieur de la procédure et elles perdent leur valeur entre les appels.

Private

Variables visibles uniquement à l'intérieur du script dans lequel elles sont déclarées.

---

procédure

Séquence nommée d'instructions exécutée comme une unité. Par exemple, Function et Sub sont des types de procédures.

---

propriété

Attribut nommé d'objet. Les propriétés définissent les caractéristiques de l'objet telles que la taille, la couleur, l'emplacement sur l'écran ou l'état d'un objet (activé ou désactivé, par exemple).

Public

Variables déclarées avec l'instruction Public ; elles sont visibles par toutes les procédures de tous les modules de toutes les applications.

SCODE

Entier long utilisé pour transmettre des informations détaillées à l'appelant d'un membre d'interface ou d'une fonction API. Les codes d'état des interfaces OLE et des API sont définis dans FACILITY_ITF.

séparateurs de date

Caractères utilisés pour séparer le jour, le mois et l'année quand les valeurs de date sont mises en forme.

---

tableau

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.

---

type d'objet

Type d'objet exposé par une application, par exemple, Application, File, Range et Sheet. Pour obtenir une liste complète des objets disponibles, reportez-vous à la documentation de l'application (Microsoft Excel, Microsoft Project, Microsoft Word, etc.).

---

valeur initiale

Valeur initiale utilisée pour générer des nombres pseudo-aléatoires. Par exemple, l'instruction Randomize crée une valeur initiale utilisée par la fonction Rnd afin de générer des séquences de nombres pseudo-aléatoires uniques.

---

variable

Emplacement de stockage nommé qui peut contenir des données pouvant être modifiées pendant l'exécution du programme. Chaque variable a un nom qui l'identifie de façon unique à l'intérieur de son niveau de portée.

Les noms de variables :

- doivent commencer par un caractère alphabétique ;
- ne peuvent contenir une virgule intégrée ou un caractère de déclaration de type ;
- doivent être uniques à l'intérieur de la même portée ;
- ne doivent pas dépasser 255 caractères.

Microsoft® Visual Basic® Scripting Edition

Référence du langage

CBool, fonction

Voir aussi

Description

Retourne une expression qui a été convertie en un Variant de sous-type Boolean.

Syntaxe

CBool(*expression*)

L'argument *expression* est toute expression valide.

Note

Si *expression* correspond à zéro, la valeur False est retournée ; si tel n'est pas le cas, la valeur True est retournée. Si l'argument *expression* ne peut être interprété comme valeur numérique, une erreur d'exécution se produit.

CBool, fonction

Voir aussi

CByte, fonction
CCur, fonction
CDate, fonction
CDbl, fonction
CInt, fonction
CLng, fonction
CSng, fonction
CStr, fonction

Microsoft® Visual Basic® Scripting Edition

Cbyte, fonction

Voir aussi

Description

Retourne une expression qui a été convertie en un Variant de sous-type Byte.

Syntaxe

CByte(*expression*)

L'argument *expression* représente toute expression valide.

Notes

En général, vous pouvez documenter votre code à l'aide des fonctions de conversion des sous-types pour indiquer que le résultat d'une certaine opération doit être exprimé sous forme d'un type de données particulier plutôt que sous la forme du type de données par défaut. Par exemple, utilisez CByte pour forcer l'arithmétique octet dans les cas où l'arithmétique monétaire, en simple précision, en double précision ou en nombre entier serait normalement utilisée.

Utilisez la fonction CByte pour effectuer des conversions reconnues au niveau international de tout autre type de données en sous-type Byte. Par exemple, différents séparateurs décimaux sont correctement reconnus selon les paramètres régionaux de votre système, comme le sont les différents séparateurs de milliers.

Si l'argument *expression* n'est pas compris dans la plage acceptable pour le sous-type Byte, une erreur se produit.

Cparte, fonction

Voir aussi

CBool, fonction
CCur, fonction
CDate, fonction
CDbl, fonction
CInt, fonction
CLng, fonction
CSng, fonction
CStr, fonction

Microsoft® Visual Basic® Scripting Edition

Référence du langage

CCur, fonction

Voir aussi

Description

Retourne une expression qui a été convertie en un Variant de sous-type Currency.

Syntaxe

CCur(*expression*)

L'argument *expression* correspond à n'importe quelle expression valide.

Notes

En général, vous pouvez documenter votre code en utilisant les fonctions de conversion de sous-type pour indiquer que le résultat de certaines opérations doit être exprimé sous la forme d'un type de données particulier et non sous la forme du type de données par défaut. Par exemple, utilisez la fonction CCur pour forcer l'emploi d'une arithmétique monétaire lorsqu'une arithmétique entière devrait normalement être utilisée.

Il convient d'employer la fonction CCur pour convertir d'autres types de données en un sous-type Currency tout en respectant les conventions internationales. Par exemple différents séparateurs décimaux et séparateurs de milliers sont correctement reconnus en fonction des paramètres régionaux de votre système.

Microsoft® Visual Basic® Scripting Edition

Référence du langage

CCur, fonction

Voir aussi

CBool, fonction
CByte, fonction
CDate, fonction
CDbl, fonction
CInt, fonction
CLng, fonction
CSng, fonction
CStr, fonction

Microsoft® Visual Basic® Scripting Edition

Référence du langage

CDate, fonction

Voir aussi

Description

Retourne une expression qui a été convertie en un Variant de sous-type Date.

Syntaxe

CDate(*date*)

L'argument *date* représente toute expression date valide.

Notes

Utilisez la fonction IsDate pour déterminer si l'argument *date* peut être converti en date ou en heure. CDate reconnaît les littéraux de date et heure ainsi que certains nombres compris dans la plage des dates acceptables. Lors de la conversion d'un nombre en date, la partie entière du nombre est convertie en date. Toute partie fractionnaire du nombre est convertie en heure du jour, commençant à minuit.

CDate reconnaît les formats de date en fonction des paramètres régionaux de votre système. L'ordre correct du jour, du mois et de l'année ne peut être déterminé s'il est fourni dans un format différent de celui reconnu par votre paramétrage de date. Par ailleurs, un format de date de type long n'est pas reconnu s'il contient aussi la chaîne jour de la semaine.

Microsoft® Visual Basic® Scripting Edition

CDate, fonction

Voir aussi

IsDate, fonction

Microsoft® Visual Basic® Scripting Edition

IsDate, fonction

Voir aussi

Description

Retourne une valeur booléenne indiquant si une expression peut être convertie en date.

Syntaxe

IsDate(*expression*)

L'argument *expression* représente toute date ou expression de chaîne reconnaissable sous forme de date ou d'heure.

Notes

La fonction IsDate retourne la valeur True si l'expression est une date ou si elle peut être convertie en date valide; si tel n'est pas le cas, elle retourne la valeur False. Dans Microsoft Windows, la plage des dates valides est comprise entre le 1er janvier 100 (après J.C.) et le 31 décembre 9999 (après J.C.); cette plage varie selon les systèmes d'exploitation.

Microsoft® Visual Basic® Scripting Edition

Référence du langage

IsDate, fonction

Voir aussi

CDate, fonction
IsArray, fonction
IsEmpty, fonction
IsNull, fonction
IsNumeric, fonction
IsObject, fonction
VarType, fonction

IsArray, fonction

Voir aussi

Description

Retourne une valeur booléenne indiquant si la variable est un tableau.

Syntaxe

IsArray(*varname*)

L'argument *varname* représente toute variable.

Notes

La fonction IsArray retourne la valeur True si la variable est un tableau; si tel n'est pas le cas, elle retourne la valeur False. La fonction IsArray est particulièrement utile avec des variants contenant des tableaux.

Microsoft® Visual Basic® Scripting Edition

Référence du langage

IsArray, fonction

Voir aussi

IsDate, fonction
IsEmpty, fonction
IsNull, fonction
IsNumeric, fonction
IsObject, fonction
VarType, fonction

Microsoft® Visual Basic® Scripting Edition

IsEmpty, fonction

Description

Retourne une valeur booléenne indiquant si une variable a été initialisée.

Syntaxe

IsEmpty(*expression*)

L'argument *expression* représente toute expression. Cependant, dans la mesure où la fonction IsEmpty est utilisée pour déterminer si des variables individuelles sont initialisées, l'argument *expression* est très souvent un nom de variable simple.

Note

La fonction IsEmpty retourne la valeur True si la variable n'est pas initialisée ou explicitement définie sur Empty; si tel n'est pas le cas, elle retourne la valeur False. La valeur False est toujours retournée si l'argument *expression* contient plusieurs variables.

Microsoft® Visual Basic® Scripting Edition

Référence du langage

IsEmpty, fonction

Voir aussi

IsArray, fonction
IsDate, fonction
IsNull, fonction
IsNumeric, fonction
IsObject, fonction
VarType, fonction

Microsoft® Visual Basic® Scripting Edition

IsNull, fonction

Voir aussi

Description

Retourne une valeur booléenne indiquant si une expression contient des données valides ou non (Null).

Syntaxe

IsNull(*expression*)

L'argument *expression* représente toute expression.

Notes

La fonction IsNull retourne la valeur True si l'argument *expression* est Null, autrement dit s'il ne contient aucune donnée valide ; dans le cas contraire, la fonction IsNull retourne la valeur False. Si l'argument *expression* se compose de plusieurs variables, la présence de la valeur Null dans toute variable constituante provoque le retourne de la valeur True pour l'expression entière.

La valeur Null indique que la variable ne contient aucune donnée valide. La valeur Null est différente de la valeur Empty qui indique qu'une variable n'a pas encore été initialisée. Elle diffère également d'une chaîne de longueur nulle que l'on appelle parfois chaîne vide.

Important Utilisez la fonction **IsNull** pour déterminer si une expression contient une valeur **Null**. Les expressions qui donneront vraisemblablement comme résultat **True** dans certaines circonstances, telles que If Var = Null et If Var <> Null, sont toujours **False**, car toute expression contenant une valeur **Null** est elle-même **Null** et donc **False**.

Microsoft® Visual Basic® Scripting Edition

Référence du langage

IsNull, fonction

Voir aussi

IsArray, fonction
IsDate, fonction
IsEmpty, fonction
IsNumeric, fonction
IsObject, fonction
VarType, fonction

IsNumeric, fonction

Voir aussi

Description

Retourne une valeur booléenne indiquant si une expression peut être évaluée sous la forme d'un nombre.

Syntaxe

IsNumeric(*expression*)

L'argument *expression* représente toute expression.

Notes

La fonction IsNumeric retourne la valeur True si l'argument *expression* entier est reconnu comme un nombre; dans le cas contraire, elle retourne la valeur False.

La fonction IsNumeric retourne la valeur False si l'argument *expression* est une expression de date.

Microsoft® Visual Basic® Scripting Edition

Référence du langage

IsNumeric, fonction

Voir aussi

IsArray, fonction
IsDate, fonction
IsEmpty, fonction
IsNull, fonction
IsObject, fonction
VarType, fonction

Microsoft® Visual Basic® Scripting Edition

IsObject, fonction

Voir aussi

Description

Retourne une valeur booléenne indiquant si une expression référence un objet Automation valide.

Syntaxe

IsObject(*expression*)

L'argument *expression* représente toute expression.

Note

La fonction IsObject retourne la valeur True si l'argument *expression* est une variable de sous-type Object ou un objet défini par l'utilisateur; si ce n'est pas le cas, elle retourne la valeur False.

Microsoft® Visual Basic® Scripting Edition

Référence du langage

IsObject, fonction

Voir aussi

IsArray, fonction
IsDate, fonction
IsEmpty, fonction
IsNull, fonction
IsNumeric, fonction
Set, instruction
VarType, fonction

Microsoft® Visual Basic® Scripting Edition

Set, instruction

Voir aussi

Description

Affecte une référence d'objet à une variable ou à une propriété.

Syntaxe

Set *objectvar* = { *objectexpression* | Nothing}

La syntaxe de l'instruction Set comporte les éléments suivants :

Élément	Description
objectvar	Nom de la variable ou propriété ; respecte les conventions standard d'affectation de nom à des variables.
objectexpression	Expression composée du nom d'un objet, d'une autre variable déclarée du même type d'objet ou d'une fonction ou méthode qui retourne un objet appartenant au même type d'objet.
Nothing	Met fin à l'association de l'élément *objectvar* à un objet spécifique. L'affectation à l'élément *objectvar* de la valeur **Nothing** libère toutes les ressources système et mémoire associées à l'objet précédemment référencé quand aucun autre élément n'y fait référence.

Notes

Pour être valide, *objectvar* doit être un type d'objet en cohérence avec l'objet qui lui est affecté.

Les instructions Dim, Private, Public ou ReDim ne déclarent qu'une variable faisant référence à un objet. Il n'est fait référence à aucun objet réel avant que vous n'utilisiez l'instruction Set pour affecter un objet spécifique.

En général, quand vous utilisez Set pour affecter une référence d'objet à une variable, aucune copie de l'objet n'est créée pour cette variable. À la place, une référence à l'objet est créée. Plusieurs variables objets peuvent faire référence au même objet. Dans la mesure où ces variables sont des références à l'objet (plutôt que des copies), tout changement apporté à l'objet est répercuté dans toutes les variables y faisant référence.

Microsoft® Visual Basic® Scripting Edition

Référence du langage

Set, instruction

Voir aussi

Dim, instruction
ReDim, instruction

Microsoft® Visual Basic® Scripting Edition

Dim, instruction

Voir aussi

Description

Déclare des variables et alloue l'espace de stockage.

Syntaxe

Dim *varname*[([*subscripts*])][, *varname*[([*subscripts*])]] . . .

La syntaxe de l'instruction Dim comprend les éléments suivants :

Élément	Description
varname	Nom de la variable; respecte les conventions standard d'affectation de nom à des variables.
subscripts	Dimensions d'une variable tableau ; jusqu'à 60 dimensions multiples peuvent être déclarées. L'argument *subscripts* utilise la syntaxe suivante : *upperbound* [**,***upperbound*] **. . .** La limite inférieure d'un tableau est toujours zéro.

Notes

Les variables déclarées avec Dim au niveau du script sont disponibles pour toutes les procédures contenues dans le script. Les variables de niveau procédure, ne sont disponibles que dans la procédure.

Vous pouvez aussi utiliser l'instruction Dim avec des parenthèses vides pour déclarer un tableau dynamique. Une fois cette déclaration effectuée, utilisez l'instruction ReDim à l'intérieur d'une procédure pour définir le nombre de dimensions et d'éléments contenus dans le tableau. Si vous essayez de déclarer de nouveau une dimension pour une variable tableau dont la taille a été spécifiée explicitement dans une instruction Dim, une erreur se produit.

Quand des variables sont initialisées, une variable numérique est initialisée sur 0 et une chaîne est initialisée sur une chaîne de longueur nulle ("").

Conseil : Si vous utilisez l'instruction **Dim** à l'intérieur d'une procédure, il est couramment accepté, dans la pratique générale de programmation, de placer l'instruction **Dim** au commencement de la procédure.

Microsoft® Visual Basic® Scripting Edition

Référence du langage

Dim, instruction

Voir aussi

Private, instruction
Public, instruction
ReDim, instruction
Set, instruction

Microsoft® Visual Basic® Scripting Edition

Private, instruction

Voir aussi

Description

Utilisée au niveau script pour déclarer des variables privées et allouer de l'espace de stockage.

Syntaxe

Private *varname*[([*subscripts*])][, *varname*[([*subscripts*])]] . . .

La syntaxe de l'instruction Private comprend les éléments suivants:

Élément	Description
varname	Nom de la variable; respecte les conventions standard d'attribution de nom de variable.
subscripts	Dimensions d'une variable de tableau; jusqu'à 60 dimensions peuvent être déclarées. L'argument *subscripts* utilise la syntaxe suivante: *upper* [**,** *upper*] **. . .** La limite inférieure d'un tableau a toujours la valeur zéro.

Notes

Les variables Private sont accessibles uniquement dans le script où elles ont été déclarées.

Une variable se référant à un objet doit être affectée à un objet existant à l'aide de l'instruction Set avant de pouvoir être utilisée. Jusqu'à ce qu'elle ait été affectée à un objet, la variable objet déclarée a la valeur spéciale Nothing.

Vous pouvez également employer l'instruction Private avec des parenthèses vides pour déclarer un tableau dynamique. Après la déclaration d'un tableau dynamique, utilisez l'instruction ReDim dans une procédure pour définir le nombre de dimensions et d'éléments du tableau. Si vous tentez de redéclarer une dimension pour une variable de tableau dont la taille a été explicitement spécifiée dans une instruction Private, Public ou Dim, une erreur se produit.

Lors de l'initialisation de variables, une variable de type numérique est initialisée à 0 et une variable de type String est initialisée avec une chaîne de longueur nulle ("").

Conseil Il convient de placer l'instruction **Private** au début de la procédure lorsque vous l'utilisez.

Microsoft® Visual Basic® Scripting Edition

Référence du langage

Private, instruction

Voir aussi

Dim, instruction
Function, instruction
Public, instruction
ReDim, instruction
Set, instruction
Sub, instruction

Microsoft® Visual Basic® Scripting Edition

Function, instruction

Voir aussi

Description

Déclare le nom, les arguments et le code qui forment le corps d'une procédure Function.

Syntaxe

```
[Public | Private]Function name [(arglist)]
   [statements]
   [name = expression]
   [Exit Function]
   [statements]
   [name = expression]
End Function
```

La syntaxe de l'instruction Function comprend les éléments suivants:

Élément	Description
Public	Indique que la procédure **Function** est accessible à toutes les autres procédures dans tous les scripts.
Private	Indique que la procédure **Function** est accessible uniquement aux autres procédures du script dans lequel elle est déclarée.
name	Nom de la procédure **Function**; respecte les conventions standard d'affectation de nom à des variables.
arglist	Liste de variables représentant les arguments qui sont transmis à la procédure **Function** lorsqu'elle est appelée. Plusieurs variables sont séparées par des virgules.
statements	Tout groupe d'instructions à exécuter dans le corps de la procédure **Function**.
expression	Valeur de retour de la procédure **Function**.

L'argument *arglist* comporte la syntaxe et les éléments suivants:

[ByVal | ByRef] *varname*[( )]

Élément	Description
ByVal	Indique que l'argument est transmis par valeur.
ByRef	Indique que l'argument est transmis par référence.
varname	Nom de la variable représentant l'argument qui respecte les conventions standard d'affectation de nom à des variables.

Notes

En l'absence de spécification explicite par l'intermédiaire de Public ou Private, les procédures Function sont publiques, autrement dit, elles sont visibles pour toutes les autres procédures de votre script. La valeur des variables locales dans une Function n'est pas conservée entre les appels à la procédure.

L'ensemble du code exécutable doit être contenu dans des procédures. Vous ne pouvez pas définir une procédure Function à l'intérieur d'une autre procédure Function ou Sub.

L'instruction Exit Function provoque la sortie immédiate d'une procédure Function. L'exécution du programme se poursuit avec l'instruction succédant à l'instruction ayant appelé la procédure Function. Il n'existe pas de limite au nombre d'instructions Exit Function pouvant apparaître n'importe où dans une procédure Function.

À l'instar d'une procédure Sub, une procédure Function est une procédure distincte qui peut prendre des arguments, exécuter une série d'instructions et changer la valeur de ses arguments. Toutefois, contrairement à une procédure Sub, vous pouvez utiliser une procédure Function sur le côté droit d'une expression de la même manière que vous utilisez une fonction intrinsèque comme Sqr, Cos ou Chr, quand vous voulez utiliser la valeur retournée par la fonction.

Vous appelez une procédure Function en utilisant le nom de fonction, suivi de la liste des arguments entre parenthèses, dans une expression. Pour toute information spécifique sur la manière d'appeler les procédures Function, consultez l'instruction Call.

Attention Les procédures **Function** peuvent être récursives; autrement dit, elles peuvent s'appeler elles-mêmes pour effectuer une tâche donnée. Toutefois la récursivité peut amener au dépassement de la capacité de la pile.

Pour retourner une valeur à partir d'une fonction, affectez cette valeur au nom de la fonction. Il n'existe pas de limite au nombre de ces affectations pouvant apparaître n'importe où dans la procédure. Si aucune valeur n'est affectée à *name*, la procédure retourne une valeur par défaut: une fonction numérique retourne 0 et une fonction chaîne retourne une chaîne de longueur nulle (""). Une fonction qui retourne une référence d'objet retourne la valeur Nothing si aucune référence d'objet n'est affectée à *name* (en utilisant Set) dans la Function.

L'exemple suivant décrit la manière d'affecter une valeur de retour à une fonction nommée BinarySearch. Dans ce cas, la valeur False est affectée au nom pour indiquer qu'une certaine valeur n'a pas été trouvée.

```
Function BinarySearch(. . .)
    . . .
    ' Valeur non trouvée. Retourne la valeur False.
    If lower > upper Then
        BinarySearch = False
        Exit Function
    End If
    . . .
End Function
```

Les variables utilisées dans les procédures Function se divisent en deux catégories: celles explicitement déclarées dans la procédure et celles qui ne le sont pas. Les premières (déclarées en utilisant Dim ou l'équivalent) sont toujours locales pour la procédure. Les variables qui sont utilisées mais qui ne sont pas explicitement déclarées dans une procédure sont également locales à moins qu'elles n'aient été explicitement déclarées à un niveau supérieur hors de la procédure.

Attention Une procédure peut utiliser une variable qui n'est pas déclarée explicitement dans la procédure, mais un conflit peut se produire si tout élément que vous avez défini au niveau du script porte le même nom que la variable. Si votre procédure fait référence à une variable non déclarée qui porte le même nom qu'une autre procédure, constante ou variable, il est supposé que votre procédure fait référence au nom situé au niveau du script. Déclarez explicitement les variables pour éviter ce type de conflit. Vous pouvez utiliser une instruction **Option Explicit** pour forcer la déclaration explicite des variables.

Attention VBScript peut réorganiser les expressions arithmétiques de manière à optimiser l'efficacité interne. Évitez d'utiliser une procédure **Function** dans une expression arithmétique quand la fonction change la valeur des variables dans la même expression.

Function, instruction

Voir aussi

Call, instruction
Dim, instruction
Nothing
Set, instruction
Sub, instruction

Microsoft® Visual Basic® Scripting Edition

Call, instruction

Description

Transfère le contrôle à une procédure Sub ou à une procédure Function.

Syntaxe

[Call] *name* [*argumentlist*]

La syntaxe de l'instruction Call comprend les éléments suivants :

Élément	Description
Call	Mot clé facultatif ; si spécifié, vous devez mettre *argumentlist* entre parenthèses. Par exemple : `Call MyProc(0)`
name	Nom de la procédure d'appel.
argumentlist	Liste, délimitée par des virgules, de variables, de tableaux ou d'expressions transmises à la procédure.

Notes

Vous n'êtes pas obligé d'utiliser le mot clé Call quand vous appelez une procédure. Toutefois, si vous utilisez le mot clé Call pour appeler une procédure exigeant des arguments, *argumentlist* doit être placé entre parenthèses. Si vous omettez le mot clé Call, vous devez aussi omettre les parenthèses délimitant *argumentlist*. Si vous utilisez la syntaxe Call pour appeler toute fonction intrinsèque ou définie par l'utilisateur, la valeur de retour de la fonction est ignorée.

Microsoft® Visual Basic® Scripting Edition

Référence du langage

Nothing

Voir aussi

Description

Le mot clé Nothing dans VBScript est utilisé pour dissocier une variable objet de l'objet réel. Utilisez l'instruction Set pour affecter Nothing à une variable objet. Par exemple:

```
Set MyObject = Nothing
```

Plusieurs variables objets peuvent faire référence au même objet réel. Quand Nothing est affecté à une variable objet, celle-ci ne fait plus référence à un objet réel. Quand plusieurs variables objets font référence au même objet, les ressources mémoire et système associées à l'objet référencé par les variables ne sont libérées qu'une fois que toutes les variables ont été définies sur Nothing, explicitement en utilisant Set, ou implicitement après que la dernière variable objet définie sur Nothing soit hors de portée.

Nothing

Voir aussi

Dim, instruction
Set, instruction

Microsoft® Visual Basic® Scripting Edition

ReDim, instruction

Voir aussi

Description

Utilisée au niveau de la procédure pour déclarer des variables de tableaux dynamiques, ainsi que pour allouer et réallouer l'espace de stockage.

Syntaxe

ReDim [Preserve] *varname*(*subscripts*) [, *varname*(*subscripts*)] . . .

La syntaxe de l'argument ReDim comprend les éléments suivants :

Élément	Description
Preserve	Conserve les données d'un tableau existant quand vous changez la taille de la dernière dimension.
varname	Nom de la variable ; respecte les conventions standard d'affectation de nom à des variables.
subscripts	Dimensions d'une variable d'un tableau ; jusqu'à 60 dimensions multiples peuvent être déclarées. L'argument *subscripts* utilise la syntaxe suivante : *upper* [**,***upper*] **. . .**

La limite inférieure d'un tableau est toujours zéro.

Notes

L'instruction ReDim est utilisée pour dimensionner ou redimensionner un tableau dynamique qui a déjà été déclaré formellement en utilisant une instruction Private, Public ou Dim avec des parenthèses vides (sans indice de dimension).

Vous pouvez utiliser l'instruction ReDim de façon itérative pour changer le nombre d'éléments et les dimensions d'un tableau.

Si vous utilisez le mot clé Preserve, vous ne pouvez modifier que la dernière dimension du tableau et, en aucun cas, le nombre de dimensions. Par exemple, si votre tableau ne comporte qu'une seule dimension, vous pouvez la modifier car c'est la dernière et seule dimension. Toutefois, si votre tableau comporte deux ou plusieurs dimensions, vous ne pouvez modifier que la dernière dimension, tout en conservant le contenu du tableau. L'exemple suivant montre comment vous pouvez augmenter la taille de la dernière dimension d'un tableau dynamique, sans pour autant effacer les données contenues dans ce dernier.

```
ReDim X(10, 10, 10)
. . .
ReDim Preserve X(10, 10, 15)
```

Attention Si vous réduisez la taille originale d'un tableau, les données contenues dans les éléments éliminés sont perdues.

Quand les variables sont initialisées, une variable numérique est initialisée à 0 et une variable de chaîne est initialisée avec une chaîne de longueur nulle (""). Une variable faisant référence à un objet doit être affectée à un objet existant à l'aide de l'instruction Set avant de pouvoir être utilisée. Jusqu'à ce qu'elle soit affectée à un objet, la variable objet déclarée possède la valeur spéciale Nothing.

Microsoft® Visual Basic® Scripting Edition

Référence du langage

ReDim, instruction

Voir aussi

Dim, instruction
Set, instruction

Microsoft® Visual Basic® Scripting Edition

Sub, instruction

Voir aussi

Description

Déclare le nom, les arguments et le code qui forment le corps d'une procédure Sub

Syntaxe

```
[Public | Private] Sub name [(arglist)]
   [statements]
   [Exit Sub]
   [statements]
End Sub
```

La syntaxe de l'instruction Sub comprend les éléments suivants :

Élément	Description
Public	Indique que la procédure **Sub** est accessible à toutes les autres procédures dans tous les scripts.
Private	Indique que la procédure **Sub** est accessible uniquement aux autres procédures du script dans lequel elle est déclarée.
name	Nom de la procédure **Sub**; respecte les conventions standard d'affectation de nom à des variables.
arglist	Liste de variables représentant les arguments passés à la procédure **Sub** quand elle est appelée. Les variables multiples sont séparées par des virgules.
statements	Tout groupe d'instructions à exécuter dans le corps de la procédure **Sub**.

L'argument *arglist* comprend la syntaxe et les éléments suivants :

[ByVal | ByVal] *varname*[( )]

Élément	Description
ByVal	Indique que l'argument est passé par valeur.
ByRef	Indique que l'argument est passé par référence.
varname	Nom de la variable représentant l'argument ; respecte les conventions standard d'affectation de nom à des variables.

Notes

En l'absence des mots clés Public ou Private, les procédures Sub sont publiques par défaut. En d'autres termes, elles sont visibles pour toutes les autres procédures de votre script. La valeur des variables locales dans une procédure Sub n'est pas conservée entre les appels à la procédure.

L'ensemble du code exécutable doit être contenu dans des procédures. Vous ne pouvez définir une procédure Sub à l'intérieur d'une autre procédure Sub ou Function.

L'instruction Exit Sub provoque la sortie immédiate d'une procédure Sub. L'exécution du programme se poursuit avec l'instruction suivant l'instruction ayant appelé la procédure Sub. Une procédure Sub peut comporter un nombre indéterminé d'instructions Exit Sub apparaissant en n'importe quel point.

À l'instar d'une procédure Function, une procédure Sub est une procédure distincte qui peut prendre des arguments, exécuter une série d'instructions et changer la valeur de ses arguments. Toutefois, contrairement à la procédure Function qui retourne une valeur, une procédure Sub ne peut pas être utilisée dans une expression.

Vous appelez une procédure Sub en utilisant le nom de la procédure, suivi de la liste des arguments. Pour plus d'informations sur la manière d'appeler les procédures Sub, consultez l'instruction Call.

Attention Les procédures **Sub** peuvent être récursives ; autrement dit, elles peuvent s'appeler elles-mêmes pour effectuer une tâche donnée. Toutefois, la récursivité peut amener au dépassement de la capacité de la pile.

Les variables utilisées dans les procédures Sub se divisent en deux catégories : celles qui sont déclarées explicitement dans la procédure et celles qui ne le sont pas. Les premières (déclarées en utilisant Dim ou l'équivalent) sont toujours locales pour la procédure. Les variables utilisées sans avoir été déclarées explicitement dans une procédure sont toujours locales, à moins qu'elles n'aient été explicitement déclarées à un niveau supérieur hors de la procédure.

Attention Une procédure peut utiliser une variable qui n'est pas déclarée explicitement dans la procédure, mais un conflit peut se produire si vous avez défini un élément au niveau du script qui porte le même nom que celui de la variable. Si votre procédure fait référence à une variable non déclarée portant le même nom qu'une autre procédure, constante ou variable, il est supposé que votre procédure fait référence à ce nom au niveau du script. Déclarez explicitement les variables pour éviter ce type de conflit. Vous pouvez utiliser une instruction **Option Explicit** pour forcer la déclaration explicite des variables.

Microsoft® Visual Basic® Scripting Edition

Référence du langage

Sub, instruction

Voir aussi

Call, instruction
Dim, instruction
Function, instruction

Microsoft® Visual Basic® Scripting Edition

Public, instruction

Voir aussi

Description

Utilisée au niveau script pour déclarer des variables publiques et allouer de l'espace de stockage.

Syntaxe

Public *varname*[([*subscripts*])][, *varname*[([*subscripts*])]] . . .

La syntaxe de l'instruction Public comprend les éléments suivants :

Élément	Description
varname	Nom de la variable ; respecte les conventions standard d'attribution de nom de variable.
subscripts	Dimensions d'une variable de tableau ; jusqu'à 60 dimensions peuvent être déclarées. Syntaxe de l'argument *subscripts* : *upper* [,*upper*] . . . La limite inférieure d'un tableau a toujours la valeur zéro.

Notes

Les variables déclarées avec l'instruction Public sont accessibles dans toutes les procédures de tous les scripts de tous les projets.

Une variable se référant à un objet doit être affectée à un objet existant à l'aide de l'instruction Set avant de pouvoir être utilisée. Jusqu'à ce qu'elle ait été affectée à un objet, la variable objet déclarée a la valeur spéciale Nothing.

Vous pouvez également employer l'instruction Public avec des parenthèses vides pour déclarer un tableau dynamique. Après la déclaration d'un tableau dynamique, utilisez l'instruction ReDim dans une procédure pour définir le nombre de dimensions et d'éléments du tableau. Si vous tentez de redéclarer une dimension pour une variable de tableau dont la taille a été explicitement spécifiée dans une instruction Private, Public ou Dim, une erreur se produit.

Lors de l'initialisation de variables, une variable de type numérique est initialisée à 0 et une variable de type String est initialisée avec une chaîne de longueur nulle ("").

Public, instruction

Voir aussi

Dim, instruction
Private, instruction
ReDim, instruction
Set, instruction

Microsoft® Visual Basic® Scripting Edition

VarType, fonction

Voir aussi

Description

Retourne une valeur indiquant le sous-type d'une variable.

Syntaxe

VarType(*varname*)

L'argument *varname* représente toute variable.

Valeurs retournées

Constante	Valeur	Description du type de variable
vbEmpty	0	**Empty** (non initialisée).
vbNull	1	**Null** (aucune donnée valide).
vbInteger	2	Entier.
vbLong	3	Entier long.
vbSingle	4	Nombre en virgule flottante en simple précision.
vbDouble	5	Nombre en virgule flottante en double précision.
vbCurrency	6	Monétaire.
vbDate	7	Date.
vbString	8	Chaîne.
vbObject	9	Objet Automation.
vbError	10	Erreur.
vbBoolean	11	Booléen.
vbVariant	12	Variant (utilisé seulement avec des tableaux de Variants).
vbDataObject	13	Objet non Automation.
vbByte	17	Octet.
vbArray	8192	Tableau.

Remarque Ces constantes sont spécifiées par VBScript. En conséquence, les noms peuvent être utilisés n'importe où dans votre code à la place des valeurs réelles.

Note

La fonction VarType ne retourne jamais la valeur du sous-type Tableau par elle-même. Elle est toujours ajoutée à une autre valeur pour indiquer un tableau d'un type particulier. La valeur du sous-type Variant n'est retournée que si elle a été ajoutée à la valeur du sous-type Tableau pour indiquer que l'argument de la fonction VarType est un tableau. Par exemple, la valeur retournée pour un tableau d'entiers est calculée comme 2 + 8192 ou 8194. Si un objet possède une propriété par défaut, la fonction VarType (*object*) retourne le type de cette propriété.

Microsoft® Visual Basic® Scripting Edition

Référence du langage

VarType, fonction

Voir aussi

IsArray, fonction
IsDate, fonction
IsEmpty, fonction
IsNull, fonction
IsNumeric, fonction
IsObject, fonction
TypeName, fonction

Microsoft® Visual Basic® Scripting Edition

TypeName, fonction

Voir aussi

Description

Retourne une chaîne qui fournit des informations de sous-type Variant sur une variable.

Syntaxe

TypeName(*varname*)

L'argument *varname* peut correspondre à n'importe quelle variable.

Valeurs retournées

La fonction TypeName retourne les valeurs suivantes:

Valeur	Description
Byte	Valeur de type octet
Integer	Valeur de type entier
Long	Valeur de type entier long
Single	Valeur en virgule flottante à simple précision
Double	Valeur en virgule flottante à double précision
Currency	Valeur de type monétaire
Decimal	Valeur de type décimale
Date	Valeur de date ou d'heure
String	Valeur de chaîne de caractères
Boolean	Valeur de type booléen ; **True** ou **False**
Empty	Non intialisée
Null	Aucune donnée valide
<object type>	Nom de type réel d'un objet
Object	Objet générique
Inconnu	Type d'objet inconnu
Nothing	Variable d'objet ne se référant encore à aucune instance d'objet
Error	Erreur

Microsoft® Visual Basic® Scripting Edition

Référence du langage

TypeName, fonction

Voir aussi

IsArray, fonction
IsDate, fonction
IsEmpty, fonction
IsNull, fonction
IsNumeric, fonction
IsObject, fonction
VarType, fonction

Microsoft® Visual Basic® Scripting Edition

Référence du langage

CDbl, fonction

Voir aussi

Description

Retourne une expression qui a été convertie en un Variant de sous-type Double.

Syntaxe

CDbl(*expression*)

L'argument *expression* représente toute expression valide.

Notes

En général, vous pouvez documenter votre code en utilisant les fonctions de conversion des sous-types pour indiquer que le résultat d'une opération doit être exprimé sous forme d'un type de données particulier plutôt que sous la forme du type de données par défaut. Par exemple, utilisez la fonction CDbl ou CSng pour forcer l'arithmétique en double ou en simple précision dans les cas où l'arithmétique monétaire ou entier serait normalement utilisée.

Utilisez la fonction CDbl pour fournir des conversions reconnues au niveau international de tout autre type de données en sous-type Double. Par exemple, différents séparateurs décimaux et séparateurs de milliers sont correctement reconnus en fonction des paramètres régionaux de votre système.

Microsoft® Visual Basic® Scripting Edition

Référence du langage

CDbl, fonction

Voir aussi

CBool, fonction
CByte Function
CCur, fonction
CDate, fonction
CInt, fonction
CLng, fonction
CSng, fonction
CStr, fonction

Microsoft® Visual Basic® Scripting Edition

Référence du langage

CInt, fonction

Voir aussi

Description

Retourne une expression qui a été convertie en un Variant de sous-type Integer.

Syntaxe

CInt(*expression*)

L'argument *expression* représente toute expression valide.

Notes

En général, vous pouvez documenter votre code en utilisant les fonctions de conversion des sous-types pour indiquer que le résultat d'une opération doit être exprimé sous forme d'un type de données particulier plutôt que sous la forme du type de données par défaut. Par exemple, utilisez la fonction CInt ou CLng pour forcer l'arithmétique entier dans les cas où l'arithmétique monétaire, en simple précision ou en double précision serait normalement utilisée.

Utilisez la fonction CInt pour fournir des conversions reconnues au niveau international de tout autre type de données en sous-type Integer. Par exemple, différents séparateurs décimaux et séparateurs des milliers sont reconnus en fonction des paramètres régionaux de votre système.

Si l'argument *expression* n'est pas compris dans la plage acceptable pour le sous-type Integer, une erreur se produit.

Remarque La fonction **CInt** est différente des fonctions **Fix** et **Int** qui tronquent au lieu d'arrondir la mantisse d'un nombre. Quand la mantisse est exactement 0,5, la fonction **CInt** l'arrondit toujours au nombre pair le plus proche. Par exemple, 0,5 est arrondi à 0 et 1,5 est arrondi à 2.

Microsoft® Visual Basic® Scripting Edition

Référence du langage

CInt, fonction

Voir aussi

CBool, fonction
CByte Function
CCur, fonction
CDate, fonction
CDbl, fonction
CLng, fonction
CSng, fonction
CStr, fonction
Int, Fix, fonctions

CLng, fonction

Voir aussi

Description

Retourne une expression qui a été convertie en un Variant de sous-type Long.

Syntaxe

CLng(*expression*)

L'argument *expression* représente toute expression valide.

Notes

En général, vous pouvez documenter votre code en utilisant les fonctions de conversion des sous-types pour indiquer que le résultat d'une opération doit être exprimé sous forme d'un type de données particulier plutôt que sous la forme du type de données par défaut. Par exemple, utilisez la fonction CInt ou CLng pour forcer l'arithmétique entier dans les cas où l'arithmétique monétaire, en simple précision ou en double précision serait normalement utilisée.

Utilisez la fonction CLng pour fournir des conversions reconnues au niveau international de tout autre type de données en sous-type Long. Par exemple, différents séparateurs décimaux et séparateurs des milliers sont correctement reconnus en fonction des paramètres régionaux de votre système.

Si l'argument *expression* n'est pas compris dans la plage acceptable pour le sous-type Long, une erreur se produit.

Remarque La fonction **CLng** est différente des fonctions **Fix** et **Int** qui tronquent au lieu d'arrondir la mantisse d'un nombre. Quand la mantisse est exactement 0,5, la fonction **CLng** l'arrondit toujours au nombre pair le plus proche. Par exemple, 0,5 est arrondi à 0 et 1,5 est arrondi à 2.

CLng, fonction

Voir aussi

CBool, fonction
CByte Function
CCur, fonction
CDate, fonction
CDbl, fonction
CInt, fonction
CSng, fonction
CStr, fonction
Int, Fix, fonctions

Microsoft® Visual Basic® Scripting Edition

Référence du langage

CSng, fonction

Voir aussi

Description

Retourne une expression qui a été convertie en un Variant de sous-type Single.

Syntaxe

CSng(*expression*)

L'argument *expression* représente toute expression valide.

Notes

En général, vous pouvez documenter votre code en utilisant les fonctions de conversion des types de données pour indiquer que le résultat d'une opération doit être exprimé sous forme d'un type de données particulier plutôt que sous la forme du type de données par défaut. Par exemple, utilisez la fonction CDbl ou CSng pour forcer l'arithmétique en double ou en simple précision dans les cas où l'arithmétique monétaire ou entier serait normalement utilisée.

Utilisez la fonction CSng pour fournir des conversions reconnues au niveau international de tout autre type de données en sous-type Single. Par exemple, différents séparateurs décimaux sont correctement reconnus en fonction des paramètres régionaux de votre système, comme les différents séparateurs de milliers.

Si l'argument *expression* n'est pas compris dans la plage acceptable pour le sous-type Single, une erreur se produit.

CSng, fonction

Voir aussi

CBool, fonction
CByte Function
CCur, fonction
CDate, fonction
CDbl, fonction
CInt, fonction
CLng, fonction
CStr, fonction

CStr, fonction

Voir aussi

Description

Retourne une expression qui a été convertie en un Variant de sous-type String.

Syntaxe

CStr(*expression*)

L'argument *expression* représente toute expression valide.

Notes

En général, vous pouvez documenter votre code en utilisant des fonctions de conversion des types de données pour indiquer que le résultat d'une opération doit être exprimé sous forme d'un type de données particulier plutôt que sous la forme du type de données par défaut. Par exemple, utilisez la fonction CStr pour forcer le résultat à être exprimé sous forme d'un sous-type String.

Vous devez utiliser la fonction CStr à la place de Str pour fournir des conversions reconnues au niveau international de tout autre type de données en sous-type String. Par exemple, différents séparateurs décimaux sont correctement reconnus en fonction des paramètres régionaux de votre système.

Les données contenues dans l'argument *expression* déterminent ce qui est retourné conformément au tableau suivant:

Si expression est	La fonction CStr retourne
Boolean	Une **Chaîne** contenant **True** ou **False**.
Date	Une **Chaîne** contenant une date dans le format court de votre système.
Null	Une erreur d'exécution.
Empty	Une **Chaîne** de longueur nulle ("").
Error	Une **Chaîne** contenant le mot Erreur suivi d'un numéro d'erreur.
Autre élément numérique	Une **Chaîne** contenant le nombre.

Microsoft® Visual Basic® Scripting Edition

Référence du langage

CStr, fonction

Voir aussi

CBool, fonction
CByte Function
CCur, fonction
CDate, fonction
CDbl, fonction
CInt, fonction
CLng, fonction
CSng, fonction

Microsoft® Visual Basic® Scripting Edition

Int, Fix, fonctions

Description

Retournent la partie entière d'un nombre.

Syntaxe

Int(*number*)

Fix(*number*)

L'argument *number* représente toute expression numérique valide. Si l'argument *number* contient Null, la valeur Null est retournée.

Notes

Les deux fonctions Int et Fix suppriment la partie fractionnaire de l'argument *number* et retourne la valeur entière résultante.

La différence entre les fonctions Int et Fix tient au fait que si l'argument *number* est négatif, la fonction Int retourne le premier entier négatif inférieur ou égal à *number*, tandis que la fonction Fix retourne le premier entier négatif supérieur ou égal à *number*. Par exemple, la fonction Int convertit -8,4 en -9, tandis que la fonction Fix convertit -8,4 en -8.

Fix(*number*) est équivalent à :

```
Sgn(number) * Int(Abs(number))
```

Microsoft® Visual Basic® Scripting Edition

Référence du langage

Int, Fix, fonctions

Voir aussi

CInt, fonction
Round, fonction

Microsoft® Visual Basic® Scripting Edition

Round, fonction

Voir aussi

Description

Retourne un nombre arrondi à un nombre spécifié de positions décimales.

Syntaxe

Round(*expression*[, *numdecimalplaces*])

La syntaxe de la fonction Round comprend les éléments suivants:

Élément	Description
expression	Expression numérique arrondie.
numdecimalplaces	Facultatif. Nombre indiquant combien de positions à la droite de la virgule sont incluses dans le nombre arrondi. Si cette valeur est omise, les entiers sont arrondis par la fonction **Round**.

Microsoft® Visual Basic® Scripting Edition

Round, fonction

Voir aussi

Int, fonctions Fix

Hex, fonction

Voir aussi

Description

Retourne une chaîne représentant la valeur hexadécimale d'un nombre.

Syntaxe

Hex(*number*)

L'argument *number* représente toute expression valide.

Notes

Si l'argument *number* n'est pas déjà un nombre entier, il est arrondi au nombre entier le plus proche avant d'être évalué.

Si *number* **est**	**Hex retourne**
Null	**Null**.
Empty	Zéro (0).
Tout autre nombre	Huit caractères hexadécimaux maximum.

Vous pouvez représenter des nombres hexadécimaux directement en les faisant précéder de &H dans la plage correcte. Par exemple, en numérotation hexadécimale, &H10 représente le nombre décimal 16.

Hex, fonction

Voir aussi

Oct, fonction

Microsoft® Visual Basic® Scripting Edition

Oct, fonction

Voir aussi

Description

Retourne une chaîne représentant la valeur octale d'un nombre.

Syntaxe

Oct(*number*)

L'argument *number* représente toute expression valide.

Notes

Si l'argument *number* n'est pas déjà un entier, il est arrondi au nombre entier le plus proche avant d'être évalué.

Si *number* **est**	**Oct retourne la valeur**
Null	**Null**.
Empty	Zéro (0).
Tout autre nombre	11 caractères octaux maximum.

Vous pouvez représenter les nombres octaux directement en faisant précéder directement de &O les nombres compris dans la plage correcte . Par exemple, &O10 est la notation octale de la décimale 8.

Oct, fonction

Voir aussi

Hex, fonction

Microsoft® Visual Basic® Scripting Edition

Constantes de VBScript

Qu'est-ce qu'une constante ?

Une constante est un nom significatif qui symbolise un nombre ou une chaîne invariable. VBScript définit un certain nombre de constantes intrinsèques. Pour d'autres informations sur ces constantes intrinsèques, reportez-vous à la Référence du langage VBScript.

Création de constantes

Vous pouvez créer dans VBScript des constantes utilisateur par l'intermédiaire de l'instruction Const. L'instruction Const vous permet de créer des constantes de chaîne ou numériques avec des noms significatifs et de leur affecter une valeur. Par exemple :

```
Const MaChaîne = "Ceci est ma chaîne."
Const MonAge = 49
```

Remarquez que la chaîne de caractères est encadrée par des guillemets (" "). Les guillemets constituent le moyen le plus évident de différencier les valeurs chaîne des valeurs numériques. Les littéraux de date et les littéraux d'heure sont encadrés par des caractères dièse (#). Par exemple :

```
Const DateFinale = #6-1-97#
```

Il est judicieux d'adopter une convention de notation pour différencier les constantes des variables. Vous évitez ainsi de tenter d'affecter une valeur à une constante au cours de l'exécution de votre script. Par exemple, vous pouvez utiliser un préfixe "vb" ou "con" pour nommer vos constantes, ou les saisir complètement en majuscules. Différencier clairement les constantes des variables élimine les confusions lorsque les scripts deviennent plus complexes.

Microsoft® Visual Basic® Scripting Edition

Opérateurs de VBScript

VBScript comprend une gamme complète d'opérateurs :opérateurs arithmétiques, opérateurs de comparaison, opérateurs de concaténation et opérateurs logiques.

Priorité des opérateurs

Lorsqu'une expression contient plusieurs opérations, chaque partie est évaluée et résolue dans un ordre prédéterminé appelé priorité des opérateurs. Vous pouvez utiliser des parenthèses pour modifier l'ordre de priorité et forcer l'évaluation de certaines parties d'une expression avant d'autres. Les opérations à l'intérieur des parenthèses sont toujours évaluées avant celles à l'extérieur. Toutefois, à l'intérieur des parenthèses, les priorités d'opérateurs reprennent leurs droits.

Lorsque des expressions contiennent des opérateurs de plusieurs catégories, les opérateurs arithmétiques sont évalués en premier, suivis des opérateurs de comparaison, puis des opérateurs logiques. Les opérateurs de comparaison ont tous la même priorité ; ils sont donc évalués de gauche à droite. Les opérateurs arithmétiques et logiques sont évalués dans l'ordre suivant :

Arithmétique		Comparaison		Logique	
Description	**Symbole**	**Description**	**Symbole**	**Description**	**Symbole**
Exponentiation	^	Égalité	=	Négation logique	Not
Négation unaire	-	Différence	<>	Conjonction logique	And
Multiplication	*	Inférieur à	<	Disjonction logique	Or
Division	/	Supérieur à	>	Exclusion logique	Xor
Division entière	\	Inférieur ou égal à	<=	Équivalence logique	Eqv
Modulo arithmétique	Mod	Supérieur ou égal à	>=	Implication logique	Imp
Addition	+	Équivalence d'objet	Is		
Soustraction	-				
Concaténation de chaînes	&				

Lorsqu'une expression contient des multiplications et des divisions, chaque opération est évaluée dans son ordre d'occurrence, de gauche à droite. De même, lorsqu'une expression contient additions et soustractions, chaque opération est évaluée dans son ordre d'occurrence, de gauche à droite.

L'opérateur de concaténation de chaîne (&) n'est pas un opérateur arithmétique mais

sa priorité est inférieure à celle des opérateurs arithmétiques et supérieure à celle des options de comparaison. L'opérateur Is est un opérateur de comparaison de référence d'objet. Il ne compare pas les objets ou leur valeur ; il détermine si deux références d'objet font référence au même objet.

Microsoft® Visual Basic® Scripting Edition

Utilisation des instructions conditionnelles

Contrôle de l'exécution du programme

Vous pouvez contrôler le déroulement de votre script à l'aide d'instructions conditionnelles et d'instructions de boucle. Les instructions conditionnelles vous permettent d'écrire du code VBScript qui prend des décisions et répète des actions. Les instructions conditionnelles suivantes sont disponibles dans VBScript :

- Instruction If...Then...Else
- Instruction Select Case

Prises de décision avec If...Then...Else

L'instruction If...Then...Else vous permet d'évaluer si une condition a pour valeur True (vraie) ou False (fausse) et, en fonction du résultat, de spécifier une ou plusieurs instructions à exécuter. En général, la condition est une expression qui utilise un opérateur de comparaison pour comparer une valeur ou une variable avec une autre. Pour des informations sur les opérateurs de comparaison, reportez-vous à Opérateurs de comparaison. Il est possible d'imbriquer les instructions If...Then...Else sur autant de niveaux que nécessaire.

Exécution d'instructions si une condition a pour valeur True (vraie)

Pour exécuter une seule instruction lorsqu'une condition a pour valeur True, utilisez la syntaxe monoligne de l'instruction If...Then...Else. L'exemple ci-dessous présente la syntaxe monoligne. Remarquez que cet exemple ne comporte pas le mot clé Else.

```
Sub MajDate()
    Dim maDate
    maDate = #2/13/95#
    If maDate < Now Then maDate = Now
End Sub
```

Pour exécuter plusieurs lignes de code, vous devez utiliser la syntaxe multiligne (ou syntaxe de bloc). Cette syntaxe comprend l'instruction End If comme le montre l'illustration ci-dessous :

```
Sub AlerterUtilisateur(valeur)
    If valeur = 0 Then
        AlertLabel.ForeColor = vbRed
        AlertLabel.Font.Bold = True
        AlertLabel.Font.Italic = True
    End If
End Sub
```

Exécution de certaines instructions si une condition a pour valeur True et exécution d'autres instructions si une condition a pour valeur False

Vous pouvez utiliser l'instruction If...Then...Else pour définir deux blocs d'instructions exécutables : un bloc à exécuter si la condition a pour valeur True, et un autre bloc à exécuter si la condition a pour valeur False.

```
Sub AlerterUtilisateur(valeur)
    If valeur = 0 Then
        AlertLabel.ForeColor = vbRed
        AlertLabel.Font.Bold = True
        AlertLabel.Font.Italic = True
    Else
        AlertLabel.Forecolor = vbBlack
        AlertLabel.Font.Bold = False
        AlertLabel.Font.Italic = False
    End If
End Sub
```

Choisir entre plusieurs alternatives

Une variante de l'instruction If...Then...Else vous permet de choisir entre plusieurs alternatives. L'ajout de clauses ElseIf étend la fonctionnalité de l'instruction If...Then...Else pour vous permettre de contrôler le déroulement du programme en fonction de différentes possibilités. Par exemple :

```
Sub AfficherValeur(value)
    If value = 0 Then
        MsgBox valeur
    ElseIf value = 1 Then
        MsgBox valeur
    ElseIf value = 2 Then
        Msgbox valeur
    Else
        Msgbox "Valeur hors limites!"
    End If
```

Vous pouvez ajouter autant de clauses ElseIf que nécessaire pour créer des alternatives. Mais une utilisation extensive des clauses ElseIf peut devenir complexe. L'instruction Select Case constitue le meilleur moyen de choisir entre plusieurs alternatives.

Prises de décision avec Select Case

La structure Select Case offre une alternative à If...Then...ElseIf pour exécuter de façon sélective un bloc d'instructions parmi plusieurs. Une instruction Select Case offre les mêmes possibilités que If...Then...Else, mais rend le code plus efficace et plus lisible.

Une structure Select Case fonctionne avec une seule expression de test, évaluée une fois, au début de la structure. Le résultat de l'expression est ensuite comparé avec les valeurs de chaque bloc Case de la structure. Lorsqu'il y a correspondance, le bloc d'instructions Case concerné s'exécute :

```
Select Case Document.Form1.CardType.Options(SelectedIndex).Text
   Case "MasterCard"
       AfficherLogoMC
       ValiderCompteMC
   Case "Visa"
       AfficherLogoVisa
```

```
            ValiderCompteVisa
        Case "American Express"
            AfficherLogoAMEXCO
            ValiderCompteAMEXCO
        Case Else
            AfficherInconnue
            Redemander
    End Select
```

Remarquez que la structure Select Case évalue l'expression une fois au début de la structure. Par contre, la structure If...Then...ElseIf peut évaluer une expression différente pour chaque instruction ElseIf. Vous pouvez remplacer une structure If...Then...ElseIf par une structure Select Case uniquement si chaque instruction ElseIf évalue la même expression.

Microsoft® Visual Basic® Scripting Edition

Boucles de répétition du code

Utilisation des boucles pour répéter du code

Les boucles vous permettent de répéter l'exécution d'un groupe d'instructions. Certaines boucles répètent les instructions jusqu'à ce qu'une condition ait pour valeur False (fausse) ; d'autres répètent les instructions jusqu'à ce qu'une condition ait pour valeur True (vraie). Il existe aussi des boucles qui répètent des instructions un nombre de fois spécifié.

Les instructions de boucle suivantes sont disponibles dans VBScript :

- Do...Loop : effectue une boucle tant qu'une condition est True ou jusqu'à ce qu'elle le devienne.
- While...Wend : effectue une boucle tant qu'une condition a pour valeur True.
- For...Next : utilise un compteur pour exécuter des instructions un nombre de fois spécifié.
- For Each...Next : répète un groupe d'instructions pour chaque élément d'une collection ou pour chaque élément d'un tableau.

Utilisation des boucles Do

Vous pouvez utiliser les instructions Do...Loop pour exécuter un bloc d'instructions un nombre de fois indéfini. Les instructions sont répétées tant qu'une condition a pour valeur True ou jusqu'à ce qu'elle ait pour valeur True.

Répétition d'instructions tant qu'une condition a pour valeur True

Utilisez le mot clé While pour contrôler une condition dans une instruction Do...Loop. Vous pouvez contrôler la condition avant d'entrer dans la boucle (comme dans l'exemple CtrlPremierWhile suivant) ou après que la boucle se soit exécutée au moins une fois (comme dans l'exemple CtrlDernierWhile suivant). Dans la procédure CtrlPremierWhile, si monNum a pour valeur 0 au lieu de 20, les instructions à l'intérieur de la boucle ne s'exécuteront jamais. Dans la procédure CtrlDernierWhile, les instructions à l'intérieur de la boucle s'exécutent une seule fois parce que la condition a déjà pour valeur False.

```
Sub CtrlPremierWhile()
    Dim compteur, monNum
    compteur = 0
    monNum = 20
    Do While monNum > 10
        monNum = monNum - 1
        compteur = compteur + 1
    Loop
    MsgBox "La boucle a effectué " & compteur & " répétitions."
End Sub

Sub CtrlDernierWhile()
    Dim compteur, monNum
    compteur = 0
```

```
        monNum = 9
        Do
            monNum = monNum - 1
            compteur = compteur + 1
        Loop While monNum > 10
        MsgBox "La boucle a effectué " & compteur & " répétitions."
    End Sub
```

Répétition d'instructions jusqu'à ce qu'une condition prenne la valeur True

Vous pouvez utiliser le mot clé Until de deux façons pour contrôler une condition dans une instruction Do...Loop. Vous pouvez contrôler la condition avant d'entrer dans la boucle (comme dans l'exemple CtrlPremierUntil suivant) ou après que la boucle se soit exécutée au moins une fois (comme dans l'exemple CtrlDernierUntil suivant). La boucle est effective tant que la condition a pour valeur False.

```
    Sub CtrlPremierUntil()
        Dim compteur, monNum
        compteur = 0
        monNum = 20
        Do Until monNum = 10
            monNum = monNum - 1
            compteur = compteur + 1
        Loop
        MsgBox "La boucle a effectué " & compteur & " répétitions."
    End Sub

    Sub CtrlDernierUntil()
        Dim compteur, monNum
        compteur = 0
        monNum = 1
        Do
            monNum = monNum + 1
            compteur = compteur + 1
        Loop Until monNum = 10
        MsgBox "La boucle a effectué " & compteur & " répétitions."
    End Sub
```

Sortie d'une instruction Do...Loop dans une boucle

Vous pouvez sortir d'une boucle Do...Loop en utilisant l'instruction Exit Do. En règle générale, vous ne sortez d'une boucle que dans des situations particulières (pour éviter une boucle infinie, par exemple). Dans ce cas, vous devez utiliser l'instruction Exit Do uniquement dans le bloc d'instructions True d'une instruction If...Then...Else. Si la condition a pour valeur False, la boucle s'exécute normalement.

Dans l'exemple ci-dessous, monNum reçoit une valeur qui crée une boucle infinie. L'instruction If...Then...Else contrôle cette condition afin de prévenir une répétition infinie.

```
    Sub ExempleDeSortie()
        Dim compteur, monNum
```

```
        compteur = 0
        monNum = 9
        Do Until monNum = 10
            monNum = monNum - 1
            compteur = compteur + 1
            If monNum < 10 Then Exit Do
        Loop
        MsgBox "La boucle a effectué " & compteur & " répétitions."
    End Sub
```

Utilisation de While...Wend

L'instruction While...Wend figure dans VBScript à l'intention des utilisateurs avertis. Cependant, en raison du manque de souplesse de While...Wend, il est recommandé d'utiliser plutôt Do...Loop.

Utilisation de For...Next

Vous pouvez utiliser les instructions For...Next pour exécuter un bloc d'instructions un nombre de fois spécifié. Pour les boucles, utilisez une variable compteur dont la valeur est augmentée ou diminuée à chaque répétition de la boucle.

Dans l'exemple ci-dessous, la procédure MaProc s'exécute 50 fois. L'instruction For spécifie la variable compteur x et ses valeurs de début et de fin. L'instruction Next incrémente la variable compteur de 1.

```
    Sub ExecMaProc50fois()
        Dim x
        For x = 1 To 50
            MaProc
        Next
    End Sub
```

Grâce au mot clé Step, vous pouvez spécifier la valeur d'incrémentation ou de décrémentation de la variable. Dans l'exemple ci-dessous, la variable compteur j est incrémentée de 2 à chaque itération de la boucle. Lorsque la boucle se termine, le total est la somme de 2, 4, 6, 8 et 10.

```
    Sub TotalDes2()
        Dim j, total
        For j = 2 To 10 Step 2
            total = total + j
        Next
        MsgBox "Le total est " & total
    End Sub
```

Pour décrémenter la variable compteur, vous utilisez une valeur de Step négative. Vous spécifiez alors une valeur de fin inférieure à la valeur de départ. Dans l'exemple ci-dessous, la variable compteur monNum est décrémentée de 2 à chaque itération de la boucle. Lorsque la boucle se termine, le total est la somme de 16, 14, 12, 10, 8, 6, 4 et 2.

```
    Sub NouveauTotal()
        Dim monNum, total
        For monNum = 16 To 2 Step -2
```

```
            total = total + monNum
        Next
        MsgBox "Le total est " & total
    End Sub
```

Vous pouvez quitter une instruction For...Next avant que le compteur atteigne sa valeur de fin en utilisant l'instruction Exit For. En règle générale, vous ne sortez d'une boucle que dans des situations particulières. En cas d'erreur par exemple, vous devez utiliser l'instruction Exit For uniquement dans le bloc d'instructions True d'une instruction If...Then...Else. Si la condition a pour valeur False, la boucle s'exécute normalement.

Utilisation de Each...Next

Une boucle For Each...Next ressemble à une boucle For...Next. Au lieu de répéter un groupe d'instructions un nombre de fois spécifié, une boucle For Each...Next le répète pour chaque élément d'une collection d'objets ou d'un tableau. Ceci s'avère particulièrement utile lorsque vous ne connaissez pas le nombre d'éléments d'une collection.

Dans l'exemple de code HTML suivant, le contenu d'un objet Dictionary est utilisé pour placer du texte dans plusieurs boîtes de texte :

```
<HTML>
<HEAD><TITLE>Feuilles et éléments</TITLE></HEAD>
<SCRIPT LANGUAGE="VBScript">
<!--
Sub cmdChanger_OnClick
   Dim d                    'Créer une variable
   Set d = CreateObject("Scripting.Dictionary")
   d.Add "0", "Athènes"     'Ajouter des clés et des éléments
   d.Add "1", "Belgrade"
   d.Add "2", "Le Caire"

   For Each I in d
       Document.frmForm.Elements(I).Value = D.Item(I)
   Next
End Sub
-->
</SCRIPT>
<BODY>
<CENTER>
<FORM NAME="frmForm"

<Input Type = "Text"><p>
<Input Type = "Text"><p>
<Input Type = "Text"><p>
<Input Type = "Text"><p>
<Input Type = "Button" NAME="cmdChanger" VALUE="Cliquez ici"><p>
</FORM>
</CENTER>
</BODY>
</HTML>
```

Microsoft® Visual Basic® Scripting Edition

Procédures de VBScript

Les types de procédures

Dans VBScript, il existe deux types de procédures : la procédure Sub et la procédure Function.

Procédures Sub

Une procédure Sub est une série d'instructions VBScript, encadrée par les instructions Sub et End Sub, qui effectue des actions mais ne retourne 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.

La procédure Sub suivante utilise deux fonctions VBScript intrinsèques, ou intégrées, MsgBox et InputBox, pour demander des informations à l'utilisateur. Elle affiche ensuite les résultats d'un calcul basé sur ces informations. Le calcul est effectué dans une procédure Function créée à l'aide de VBScript. La procédure Function est présentée dans le prochain paragraphe.

```
Sub ConvertTemp()
   temp = InputBox("Veuillez entrer la température en degrés F.", 1)
   MsgBox "La température est de " & Celsius(temp) & " degrés C."
End Sub
```

Procédures 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 retourner une valeur. Une procédure Function peut accepter des arguments (constantes, variables ou expressions transmises par une procédure appelante). Si une procédure Function n'a pas d'arguments, son instruction Function doit présenter une paire de parenthèses vide. Une procédure Function retourne une valeur en affectant une valeur à son nom dans une ou plusieurs instructions de la procédure. Le type du retour d'une Function est toujours Variant.

Dans l'exemple ci-dessous, la fonction Celsius calcule les degrés Celsius à partir des degrés Fahrenheit. Lorsque la fonction est appelée à partir de la procédure Sub ConvertTemp, une variable contenant la valeur argument lui est transmise. Le résultat du calcul est retourné à la procédure appelante et affiché dans une boîte de message.

```
Sub ConvertTemp()
    temp = InputBox("Veuillez entrer la température en degrés F.", 1)
    MsgBox "La température est de " & Celsius(temp) & " degrés C."
End Sub

Function Celsius(degrésF)
    Celsius = (degrésF - 32) * 5 / 9
End Function
```

Échange des données avec les procédures

Chaque élément de données est transmis à vos procédures par l'intermédiaire d'un argument. Les arguments servent de symboles aux données que vous voulez transmettre à la procédure. Vous pouvez nommer vos arguments de la même manière que vous nommez des variables. Lorsque vous créez une procédure par une instruction Sub ou une instruction Function, les parenthèses doivent figurer après le nom de la procédure. Les arguments éventuels figurent

entre ces parenthèses, séparés par des virgules. Par exemple, dans l'exemple suivant degrésF symbolise la valeur transmise à la fonction Celsius pour conversion :

```
Function Celsius(degrésF)
   Celsius = (degrésF - 32) * 5 / 9
End Function
```

Pour récupérer les données d'une procédure, vous devez utiliser une Function. Une procédure Function peut retourner une valeur, une procédure Sub ne le peut pas.

Utilisation des procédures Sub et Function dans le code

Dans votre code, une Function doit toujours apparaître à droite d'une affectation de variable ou dans une expression. Par exemple :

```
Temp = Celsius(degrésF)
```

ou

```
MsgBox "La température est de " & Celsius(degrésF) & " degrés C."
```

Pour appeler une procédure Sub à partir d'une autre procédure, faites simplement figurer le nom de la procédure suivi le cas échéant des valeurs arguments séparées par des virgules. L'instruction Call n'est pas obligatoire, mais si vous l'utilisez, vous devez encadrer les arguments éventuels par des parenthèses.

L'exemple ci-dessous présente deux appels à la procédure MaProc. L'un utilise l'instruction Call, l'autre non. Les deux font exactement la même chose.

```
Call MaProc(arg1, arg2)
MaProc arg1, arg2
```

Remarquez que les parenthèses sont omises lorsque l'instruction Call n'est pas utilisée.

Microsoft® Visual Basic® Scripting Edition

Conventions de codage de VBScript

Qu'entend-on par conventions de codage ?

Ces conventions sont des suggestions pouvant simplifier l'écriture de code avec Microsoft Visual Basic Scripting Edition. Elles incluent notamment :

- des conventions d'affectation de noms à des objets, des variables et des procédures,
- des conventions de commentaire,
- des directives de mise en forme et de mise en retrait.

L'utilisation de conventions cohérentes vise principalement à normaliser la structure et le style de codage d'un script ou d'un jeu de scripts pour rendre le code plus lisible et mieux compréhensible. L'emploi de conventions de codage appropriées permet de créer un code source précis, lisible, intuitif, ne présentant aucune ambiguïté, et respectant les conventions des autres langages.

Conventions d'affectation de noms à des constantes

Les versions précédentes de VBScript n'avaient pas de mécanisme de création de constantes définies par l'utilisateur. Les constantes, si elles étaient utilisées, étaient mises en œuvre comme des variables et distinguées des autres variables grâce à une syntaxe en caractères majuscules. Les différents mots étaient séparés par le caractère de soulignement (_). Par exemple :

```
MAX_LISTE_UTILISATEUR
SAUT_DE_LIGNE
```

Bien que ce mécanisme soit toujours acceptable pour identifier vos constantes, vous pouvez utiliser un nouveau schéma de notation car il est possible de créer de véritables constantes avec l'instruction Const. Cette convention utilise un format mixte, dans lequel les noms de constante commencent par le préfixe "con". Par exemple :

```
conVotrePropreConstante
```

Conventions d'affectation de noms à des variables

Pour assurer lisibilité et cohérence, utilisez dans votre code VBScript les préfixes présentés dans le tableau suivant, avec des noms de variable descriptifs.

Sous-type	Préfixe	Exemple
Boolean	bln	blnTrouvé
Byte	byt	bytDonnéesRaster
Date (Time)	dtm	dtmDébut
Double	dbl	dblTolérance
Error	err	errNumOrdre
Integer	int	intQuantité
Long	lng	lngDistance
Object	obj	objCourant
Single	sng	sngMoyenne
String	str	strPrénom

Portée des variables

La portée des variables doit toujours être la plus petite possible. Les variables VBScript peuvent avoir la portée suivante.

Portée	Endroit de déclaration de la variable	Visibilité
Niveau procédure	Procédure Event, Function ou Sub	Visible dans la procédure où elle est déclarée

Niveau script	Section HEAD d'une page HTML, hors de toute procédure	Visible dans toutes les procédures du script

Préfixes de portée de variable

Plus la taille du script augmente, plus il devient indispensable de pouvoir différencier rapidement la portée des variables. Un préfixe d'une lettre placé devant le préfixe de type assure cette différenciation, sans trop augmenter la taille des noms de variable.

Portée	Préfixe	Exemple
Niveau procédure	Aucun	dblVélocité
Niveau script	s	sblnCalculEnCours

Noms descriptifs de variable et de procédure

Le corps d'un nom de variable ou de procédure doit être composé de lettres minuscules et majuscules et décrire sa fonction. En outre, les noms de procédure doivent commencer par un verbe, tel que InitialiserTableau ou FermerDialogue.

Dans le cas de termes longs ou fréquemment utilisés, il est préférable d'employer des abréviations normalisées pour maintenir une longueur de nom raisonnable. Pour assurer la lisibilité du code, il est généralement préférable d'utiliser des noms de variable de moins de 32 caractères. Lors de l'emploi d'abréviations, assurez-vous que celles-ci sont cohérentes dans tout le script. Par exemple, l'utilisation aléatoire de Ctr et Compteur dans un script ou un ensemble de scripts peut créer une confusion.

Conventions d'affectation de noms à des objets

Le tableau suivant présente les conventions recommandées pour les divers objets utilisables en programmation VBScript.

Type d'objet	Préfixe	Exemple
Panneau 3D	pnl	pnlGroupe
Bouton animé	ani	aniBoîteALettre
Case à cocher	chk	chkLectureSeule
Liste modifiable, zone de liste déroulante	cbo	cboAnglais
Bouton de commande	cmd	cmdQuitter
Boîte de dialogue commune	dlg	dlgFichierOuvrir
Cadre	fra	fraLangage
Barre de défilement horizontale	hsb	hsbVolume
Image	img	imgIcône
Étiquette	lbl	lblMessageAide
Ligne	lin	linVerticale
Zone de liste	lst	lstCodesEmplois
Compteur	spn	spnPages
Zone de texte	txt	txtNom
Barre de défilement verticale	vsb	vsbVitesse
Curseur	sld	sldÉchelle

Conventions pour les commentaires dans le code

Toutes les procédures doivent commencer par un bref commentaire décrivant leur action. Cette description doit exclure les détails de mise en œuvre (techniques employées) parce que ceux-ci sont sujets à modification et

pourraient imposer une gestion de commentaire inutile ou pire, devenir erronés. La mise en œuvre est décrite par le code lui-même et d'éventuels commentaires sur une ligne.

Les arguments passés à une procédure doivent être décrits lorsque leur fonction n'est pas évidente et qu'ils doivent être compris dans une plage spécifique. Les valeurs de retour des fonctions et les variables modifiées par une procédure, notamment par l'intermédiaire d'arguments de référence, doivent également être décrites au début de chaque procédure.

Les commentaires des en-têtes de procédure doivent inclure les titres de section suivants. La section suivante, "Mise en forme du code", vous présente des exemples.

Titre de section	Contenu des commentaires
Objet	Ce que fait la procédure (et non comment elle le fait).
Commentaires	Liste de variables, contrôles, ou autres éléments externes dont l'état a une incidence sur cette procédure.
Effets	Présentation de l'effet de la procédure sur les variables, contrôles ou autres éléments externes.
Entrées	Explication de tous les arguments non évidents. Entrez une ligne de commentaire distincte pour chaque argument.
Valeurs retournées	Explication de la valeur retournée.

N'oubliez pas :

- Pour chaque déclaration de variable importante, prévoyez un commentaire sur une ligne décrivant l'utilisation.
- Les noms des variables, contrôles et procédures doivent être suffisamment clairs pour que les commentaires sur une ligne servent uniquement à fournir des détails de mise en œuvre complexe.
- Entrez au début de votre script une présentation générale, décrivant le script, énumérant les objets, les procédures, les algorithmes, les boîtes de dialogue et les autres dépendances système. Il peut parfois être utile d'inclure un peu de pseudocode décrivant l'algorithme.

Mise en forme du code

Bien que la mise en forme doive refléter la structure logique et l'imbrication du code, il convient d'économiser au maximum l'espace écran. Voici quelques conseils à cet égard :

- Les blocs imbriqués standard doivent être mis en retrait de quatre espaces.
- Les commentaires généraux d'une procédure doivent être mis en retrait d'un espace.
- Les instructions du plus haut niveau venant après les commentaires généraux doivent être mises en retrait de quatre espaces, chaque bloc imbriqué étant lui-même mis en retrait de quatre espaces supplémentaires. Par exemple :

```
'*******************************************************
' Objet:    Trouve la première occurrence d'un utilisateur
'           spécifié dans le tableau ListeUtilisateurs.
' Entrées:  strListeUtilisateurs(): liste des utilisateurs
'           dans laquelle effectuer la recherche.
'           strUtilisateurCible: nom de l'utilisateur à
'           rechercher.
' Retours:  L'index de la première occurrence de
'           strUtilisateurCible dans le tableau
'           ListeUtilisateurs.
'           Si l'utilisateur cible est introuvable,
'           retourne -1.
'*******************************************************

Function intChercherUtilisateur (strListeUtilisateurs(), strUtilisateurCible)
    Dim i                   ' Compteur de boucle.
    Dim blnTrouvé           ' Indicateur de cible trouvée
    intChercherUtilisateur = -1
    i = 0               ' Initialiser le compteur de boucle
    Do While i <= Ubound(strListeUtilisateurs) and Not blnTrouvé
```

```
            If strListeUtilisateurs(i) = strUtilisateurCible Then
                blnTrouvé = True   ' Indicateur à True
                intChercherUtilisateur = i ' Définir valeur retour compteur de boucle
            End If
            i = i + 1             ' Incrément compteur de boucle
        Loop
    End Function
```

Microsoft® Visual Basic® Scripting Edition

Et maintenant...

Le meilleur moyen d'apprendre les techniques VBScript avancées consiste à observer des exemples, toujours et encore des exemples. Il est également intéressant de bien connaître le modèle d'objet.

Voici des points de départ :

- FAQ sur les contrôles ActiveX™
- Exemples de pages
- Page de liens interactifs

Microsoft® Visual Basic® Scripting Edition

Une page VBScript simple

Une page simple

Avec Microsoft® Internet Explorer, vous pouvez afficher la page produite par le code HTML ci-dessous. Si vous cliquez sur le bouton de cette page, vous verrez VBScript en action.

```
<HTML>
<HEAD><TITLE>Une première page simple</TITLE>
<SCRIPT LANGUAGE="VBScript">
<!--
Sub Bouton1_OnClick
        MsgBox "Mirabile visu."
End Sub
-->
</SCRIPT>
</HEAD>
<BODY>
<H3>Une première page simple</H3><HR>
<FORM><INPUT NAME="Bouton1" TYPE="BUTTON" VALUE="Cliquez ici"></FORM>
</BODY>
</HTML>
```

Le résultat n'est pas particulièrement impressionnant : une boîte de dialogue affiche une phrase en Latin (signifiant "Remarquable à voir"). Pourtant, il se passe un certain nombre de choses.

Lorsqu'Internet Explorer lit la page, il rencontre les balises <SCRIPT>, identifie la portion de code VBScript et enregistre ce code. Lorsque vous cliquez sur le bouton, Internet Explorer relie logiquement le bouton et le code, et exécute la procédure.

La procédure Sub entre les balises <SCRIPT> est une procédure d'événement. Le nom de la procédure comprend deux parties : le nom du bouton `Bouton1` (issu de l'attribut NAME de la balise <INPUT>), et un nom d'événement `OnClick`. Les deux parties sont liées par un caractère de soulignement (_). À chaque clic sur le bouton, Internet Explorer recherche et exécute la procédure d'événement correspondante, `Bouton1_OnClick`.

Internet Explorer définit les événements disponibles pour les contrôles de la feuille dans la documentation Internet Explorer Scripting Object Model.

Les pages peuvent également utiliser des combinaisons de contrôles et de procédures. VBScript et les feuilles présente des interactions simples entre les contrôles.

Autres moyens de lier le code aux événements

Bien que la méthode ci-dessus soit probablement la plus simple et la plus générale, vous pouvez lier le code VBScript aux événements de deux autres façons. Internet Explorer vous permet d'ajouter de petites portions de code en ligne dans la balise qui définit le contrôle. Par exemple, la balise <INPUT> ci-dessous effectue la même opération que l'exemple de code précédent lorsque vous cliquez sur le bouton:

```
<INPUT NAME="Bouton1" TYPE="BUTTON"
       VALUE="Cliquez ici" OnClick='MsgBox "Mirabile visu."'>
```

Remarquez que l'appel à la fonction est encadré par des apostrophes et que la chaîne argument de la fonction MsgBox est encadrée par des guillemets. Vous pouvez utiliser

plusieurs instructions, que vous séparez par des points-virgules (:).

Vous pouvez également écrire une balise <SCRIPT> qui s'applique à un événement particulier pour un contrôle spécifique :

```
<SCRIPT LANGUAGE="VBScript" EVENT="OnClick" FOR="Bouton1">
<!--
    MsgBox "Mirabile visu."
-->
</SCRIPT>
```

Parce que la balise <SCRIPT> spécifie déjà l'événement et le contrôle, les instructions Sub et End Sub ne sont pas utilisées.

Microsoft® Visual Basic® Scripting Edition

VBScript et les feuilles

Validation simple

Vous pouvez utiliser Visual Basic Scripting Edition pour effectuer une grande partie des traitements de feuille qui nécessitent en général un serveur. Vous pouvez aussi réaliser des opérations impossibles sur un serveur.

Voici un exemple de validation simple côté client. Le code HTML correspond à un champ de texte et un bouton. Si vous utilisez Microsoft® Internet Explorer pour afficher la page produite par le code suivant, vous observerez un petit champ de texte à côté d'un bouton.

```
<HTML>
<HEAD><TITLE>Validation simple</TITLE>
<SCRIPT LANGUAGE="VBScript">
<!--
Sub Soumettre_OnClick
  Dim LaFeuille
  Set LaFeuille = Document.ValidForm
  If IsNumeric(LaFeuille.Text1.Value) Then
    If LaFeuille.Text1.Value < 1 Or LaFeuille.Text1.Value > 10 Then
      MsgBox "Veuillez entrer un nombre entre 1 et 10."
    Else
      MsgBox "Merci."
    End If
  Else
    MsgBox "Veuillez entrer une valeur numérique."
  End If
End Sub
-->
</SCRIPT>
</HEAD>
<BODY>
<H3>Validation simple</H3><HR>
<FORM NAME="ValidForm">
Entrez une valeur entre 1 et 10:
<INPUT NAME="Text1" TYPE="TEXT" SIZE="2">
<INPUT NAME="Submit" TYPE="BUTTON" VALUE="Soumettre">
</FORM>
</BODY>
</HTML>
```

La différence entre ce champ de texte et les exemples de Une page VBScript simple est que la propriété Value du champ de texte est utilisée pour vérifier la valeur entrée. Pour lire une propriété Value, le code doit qualifier la référence au nom du champ de texte.

Vous pouvez toujours écrire la référence complète `Document.FeuilleValidation.Texte1`. Toutefois, lorsque vous avez de multiples références aux contrôles de la feuille, vous emploierez la méthode suivante. Déclarez tout d'abord une variable. Utilisez ensuite l'instruction Set pour affecter la feuille à la variable `LaFeuille`. Une instruction d'affectation normale, comme Dim, ne fonctionne pas dans ce cas ; vous devez utiliser Set pour préserver la référence à un objet.

Utilisation de valeurs numériques

Remarquez que l'exemple teste directement la valeur par rapport à un nombre : il utilise la fonction IsNumeric pour vérifier que la chaîne dans le champ est un nombre. Bien que VBScript convertisse automatiquement les chaînes et les nombres, il est recommandé de tester le sous-type des valeurs saisies par l'utilisateur et d'utiliser les fonctions de conversion le cas échéant. Pour additionner des valeurs issues de champs de texte, convertissez les valeurs explicitement en nombres parce que le symbole d'opérateur + représente à la fois l'addition numérique et la concaténation de chaînes. Par exemple, si `Texte1` contient "1" et `Texte2` contient "2", on observe les résultats suivants :

```
A = Texte1.Value + Texte2.Value          ' A vaut "12"
A = CDbl(Texte1.Value) + Texte2.Value    ' A vaut 3
```

Validation et renvoi de données vers le serveur

L'exemple de validation simple utilise un simple contrôle Button. Si un contrôle Submit était utilisé, l'exemple ne verrait pas les données pour les vérifier puisqu'elles seraient envoyées immédiatement au serveur. En évitant l'utilisation du contrôle Submit, vous pouvez vérifier les données mais celles-ci ne sont pas soumises au serveur. Ceci nécessite une ligne de code supplémentaire :

```
<SCRIPT LANGUAGE="VBScript">
<!--
Sub Soumettre_OnClick
  Dim LaFeuille
  Set LaFeuille = Document.ValidForm
  If IsNumeric(LaFeuille.Texte1.Value) Then
    If LaFeuille.Texte1.Value < 1 Or LaFeuille.Texte1.Value > 10 Then
      MsgBox "Veuillez entrer un nombre entre 1 et 10."
    Else
      MsgBox "Merci."
      LaFeuille.Submit  ' Données correctes; envoyer au serveur.
    End If
  Else
    MsgBox "Veuillez entrer une valeur numérique."
  End If
End Sub
-->
</SCRIPT>
```

Pour envoyer les données au serveur, le code appelle la méthode Submit sur l'objet feuille lorsque les données sont correctes. À partir de ce moment, le serveur gère les données comme il l'aurait fait directement, sauf que les données ont été vérifiées avant d'être envoyées. Vous trouverez des informations complètes sur la méthode Submit et sur d'autres méthodes dans la page Internet Explorer Scripting Object Model.

Pour l'instant, vous avez vu uniquement les objets <FORM> HTML standard. Internet Explorer vous permet aussi d'exploiter toute la puissance des contrôles ActiveX™ (anciennement contrôles OLE) et Java™.

Microsoft® Visual Basic® Scripting Edition

Utilisation de VBScript avec les objets

Utilisation d'objets

Que vous utilisiez un contrôle ActiveX™ (anciennement nommé contrôle OLE) ou un objet Java™, Microsoft Visual Basic Scripting Edition et Microsoft® Internet Explorer le gèrent de la même façon. Si vous utilisez Internet Explorer et avez installé les contrôles disponibles dans la Galerie ActiveX, vous pouvez observer la page produite par le code ci-dessous.

Vous incorporez un objet avec des balises <OBJECT> et définissez ses valeurs de propriétés initiales avec des balises <PARAM>. Si vous êtes un programmeur Visual Basic, vous constaterez que l'utilisation des balises <PARAM> correspond à la définition des propriétés initiales d'un contrôle sur une feuille. Par exemple, l'ensemble de balises <OBJECT> et <PARAM> ajoute le contrôle Label ActiveX à une page :

```
<OBJECT
        classid="clsid:99B42120-6EC7-11CF-A6C7-00AA00A47DD2"
        id=lblActiveLbl
        width=250
        height=250
        align=left
        hspace=20
        vspace=0
>
<PARAM NAME="Angle" VALUE="90">
<PARAM NAME="Alignment" VALUE="4">
<PARAM NAME="BackStyle" VALUE="0">
<PARAM NAME="Caption" VALUE="Une simple étiquette décousue">
<PARAM NAME="FontName" VALUE="Verdana, Arial, Helvetica">
<PARAM NAME="FontSize" VALUE="20">
<PARAM NAME="FontBold" VALUE="1">
<PARAM NAME="FrColor" VALUE="0">
</OBJECT>
```

Vous pouvez lire et définir des propriétés et appeler des méthodes comme pour tout contrôle de la feuille. Le code ci-dessous, par exemple, comprend des contrôles <FORM> que vous pouvez utiliser pour manipuler deux propriétés du contrôle Label :

```
<FORM NAME="LabelControls">
<INPUT TYPE="TEXT" NAME="txtNewText" SIZE=25>
<INPUT TYPE="BUTTON" NAME="cmdChanger" VALUE="Changer le texte">
<INPUT TYPE="BUTTON" NAME="cmdPivoter" VALUE="Pivoter l'étiquette">
</FORM>
```

Lors de la définition de la feuille, une procédure d'événement pour le bouton `cmdChanger` change le texte de l'étiquette :

```
<SCRIPT LANGUAGE="VBScript">
<!--
Sub cmdChanger_onClick
        Dim LaFeuille
        Set LaFeuille = Document.LabelControls
```

```
            lblActiveLbl.Caption = LaFeuille.txtNewText.Value
    End Sub
    -->
    </SCRIPT>
```

Le code qualifie les références aux contrôles et valeurs à l'intérieur des feuilles comme dans l'exemple Validation Simple.

Plusieurs contrôles ActiveX™ sont disponibles dans la Galerie ActiveX pour utilisation avec Internet Explorer. Vous y trouverez des informations complètes sur les propriétés, méthodes et événements, ainsi que sur les identificateurs de classe (CLSID) pour les contrôles dans les pages de référence de programmation. Vous trouverez d'autres informations sur la balise <OBJECT> dans la page Internet Explorer 4.0 Author's Guide and HTML Reference.

Remarque Les versions précédentes d'Internet Explorer exigeaient des accolades ({}) autour de l'attribut d'identificateur de classe et n'étaient pas conformes à la norme W3C. L'utilisation d'accolades avec la version actuelle génère le message "This page uses an outdated version of the <OBJECT> tag".

Exemple d'objet

Cette page utilise le contrôle Label disponible dans la Galerie ActiveX™.

Validation simple

Entrez une valeur entre 1 et 10 : 

Microsoft® Visual Basic® Scripting Edition

Référence du langage

+, opérateur

Voir aussi

Description

Utilisé pour additionner deux nombres.

Syntaxe

result = *expression1* + *expression2*

La syntaxe de l'opérateur + comprend les éléments suivants :

Élément	Description
result	Toute variable numérique.
expression1	Toute expression.
expression2	Toute expression.

Notes

S'il vous est possible d'utiliser l'opérateur + pour concaténer deux chaînes de caractères, il est néanmoins préférable d'utiliser l'opérateur & pour la concaténation afin d'éliminer toute ambiguïté et de fournir un code compréhensible en lui-même.

Si vous utilisez l'opérateur +, il vous sera parfois difficile de déterminer si une addition ou une concaténation de chaîne se produira.

Le sous-type sous-jacent des expressions détermine le comportement de l'opérateur + de la manière suivante:

Si	alors
Les deux expressions sont numériques	Add.
Les deux expressions sont des chaînes	Concatenate.
Une expression est numérique et l'autre est une chaîne	Add.

Si l'une des expressions ou les deux sont Null, *result* est Null. Si les deux expressions sont Empty, *result* est un sous-type Integer. Toutefois, si une seule des expressions est Empty, l'autre expression est retournée inchangée comme *result*.

+, opérateur

Voir aussi

&, opérateur
-, opérateur
Opérateurs arithmétiques
Opérateurs de concaténation
Priorité des opérateurs
Résumé des opérateurs

Microsoft® Visual Basic® Scripting Edition

&, opérateur

Voir aussi

Description

Utilisé pour forcer la concaténation de chaînes de deux expressions.

Syntaxe

result = *expression1* & *expression2*

La syntaxe de l'opérateur & comprend les éléments suivants :

Élément	Description
result	Toute variable.
expression1	Toute expression.
expression2	Toute expression.

Note

Chaque fois qu'une *expression* n'est pas une chaîne, elle est convertie en un sous-type String. Si les deux expressions sont Null, *result* est aussi Null. Toutefois, si une seule *expression* est Null, cette expression est traitée comme une chaîne de longueur nulle lorsqu'elle est concaténée avec l'autre expression. Toute expression Empty est également traitée comme une chaîne de longueur nulle.

&, opérateur

Voir aussi

Opérateurs de concaténation
Priorité des opérateurs
Résumé des opérateurs

Opérateurs de concaténation

&, opérateur

+, opérateur

Microsoft® Visual Basic® Scripting Edition

-, opérateur

Voir aussi

Description

Utilisé pour trouver la différence entre deux nombres ou pour indiquer la valeur négative d'une expression numérique.

Syntaxe 1

result = number1-nombre2

Syntaxe 2

-number

La syntaxe de l'opérateur - comprend les éléments suivants :

Élément	Description
result	Toute variable numérique.
number	Toute expression numérique.
number1	Toute expression numérique.
number2	Toute expression numérique.

Notes

Dans la syntaxe 1, l'opérateur - est l'opérateur de soustraction arithmétique utilisé pour trouver la différence entre deux nombres. Dans la syntaxe 2, l'opérateur - est utilisé comme opérateur de négation unaire pour indiquer la valeur négative d'une expression.

Si une ou les deux expressions sont des expressions Null, *result* est Null. Si une expression est Empty, elle est traitée comme s'il s'agissait de 0.

-, opérateur

Voir aussi

+, opérateur
Opérateurs arithmétiques
Priorité des opérateurs
Résumé des opérateurs

Opérateurs arithmétiques

Microsoft® Visual Basic® Scripting Edition

^, opérateur

Voir aussi

Description

Utilisé pour élever un nombre à la puissance de l'exposant.

Syntaxe

result = *number*^*exponent*

La syntaxe de l'opérateur ^ comprend les éléments suivants :

Élément	Description
result	Toute variable numérique.
number	Toute expression numérique.
exponent	Toute expression numérique.

Notes

number ne peut être négatif que si *exponent* est une valeur en nombre entier. Quand plusieurs élévations à une puissance sont effectuées dans une même expression, l'opérateur ^ est évalué à mesure qu'il est rencontré, de gauche à droite.

Si *number* ou *exponent* sont des expressions Null, *result* est aussi Null.

^, opérateur

Voir aussi

Opérateurs arithmétiques
Priorité des opérateurs
Résumé des opérateurs

Priorité des opérateurs

Voir aussi

Description

Quand plusieurs opérations ont lieu dans une expression, chaque partie est évaluée et résolue dans un ordre prédéterminé. Cet ordre est connu sous le nom de priorité des opérateurs. Des parenthèses peuvent être utilisées pour annuler l'ordre de priorité et forcer l'évaluation de certaines parties d'une expression avant d'autres. Les opérations entre parenthèses sont toujours effectuées avant celles qui ne le sont pas. A l'intérieur des parenthèses, cependant, la priorité normale des opérateurs est conservée.

Quand des expressions contiennent des opérateurs de plusieurs catégories, les opérateurs arithmétiques sont évalués d'abord, puis les opérateurs de comparaison et enfin les opérateurs logiques. Les opérateurs de comparaison ont tous la même priorité: ils sont évalués de gauche à droite dans l'ordre de leur apparition. Les opérateurs arithmétiques et logiques sont évalués dans l'ordre de priorité suivant:

Arithmétique	**Comparaison**	**Logique**
Élévation à une puissance (^)	Égalité (=)	**Not**
Négation (-)	Inégalité (<>)	**And**
Multiplication et division (*****,/)	Inférieur à (<)	**Or**
Division des entiers (\)	Supérieur à (>)	**Xor**
Arithmétique modulo (**Mod**)	Inférieur ou (<=)	**Eqv**
Addition et soustraction (+,-)	Supérieur ou égal à (>=)	**Imp**
Concaténation de chaîne (**&**)	**Is**	&

Quand une multiplication et une division se produisent ensemble dans une expression, chaque opération est évaluée dans l'ordre de gauche à droite. De la même façon, quand une addition et une soustraction se produisent ensemble dans une expression, chaque opération est évaluée dans l'ordre d'apparition de gauche à droite.

L'opérateur de concaténation de chaîne (&) n'est pas un opérateur arithmétique mais, dans la priorité, il se situe après tous les opérateurs arithmétiques et avant tous les opérateurs de comparaison. L'opérateur Is est un opérateur de comparaison de référence d'objet. Il ne compare pas les objets ou leurs valeurs; sa vérification consiste seulement à déterminer si deux références d'objet font référence au même objet.

Priorité des opérateurs

Voir aussi

Is, opérateur
Résumé des opérateurs

Microsoft® Visual Basic® Scripting Edition

Is, opérateur

Voir aussi

Description

Utilisé pour comparer deux variables de référence à des objets.

Syntaxe

result = *object1* Is *object2*

La syntaxe de l'opérateur Is comprend les éléments suivants :

Élément	Description
result	Toute variable numérique.
object1	Tout nom d'objet.
object2	Tout nom d'objet.

Notes

Si *object1* et *object2* font tous deux référence au même objet, *result* est True; si tel n'est pas le cas, *result* est False. Il existe plusieurs manières de faire en sorte que deux variables fassent référence au même objet.

Dans l'exemple suivant, A a été défini pour faire référence au même objet que B :

```
Set A = B
```

Dans l'exemple suivant, A et B font référence au même objet que C :

```
Set A = C
Set B = C
```

Microsoft® Visual Basic® Scripting Edition

Is, opérateur

Voir aussi

Opérateurs de comparaison
Priorité des opérateurs
Résumé des opérateurs

Microsoft® Visual Basic® Scripting Edition

Opérateurs de comparaison

Voir aussi

Description

Utilisés pour comparer des expressions.

Syntaxe

result = *expression1 comparisonoperator expression2*

result = *object1* Is *object2*

result = *string* Like *pattern*

Les opérateurs de comparaison comprennent les éléments suivants :

Élément	Description
result	Toute variable numérique.
expression	Toute expression.
comparisonoperator	Tout opérateur de comparaison.
object	Tout nom d'objet.
string	Toute expression de chaîne.
pattern	Toute expression de chaîne ou série de caractères.

Note

L'opérateur Is comporte une fonctionnalité de comparaison spécifique qui diffère des autres opérateurs dans le tableau suivant. Ce tableau contient une liste des opérateurs de comparaison et des conditions qui déterminent si la valeur de *result* est True, False ou Null :

Opérateur	Description	True si	False si	Null si
<	Inférieur à	*expression1* < *expression2*	*expression1* >= *expression2*	*expression1* ou *expression2* = **Null**
<=	Inférieur ou égal à	*expression1* <= *expression2*	*expression1* > *expression2*	*expression1* ou *expression2* = **Null**
>	Supérieur à	*expression1* > *expression2*	*expression1* <= *expression2*	*expression1* ou *expression2* = **Null**
>=	Supérieur ou égal à	*expression1* >= *expression2*	*expression1* < *expression2*	*expression1* ou *expression2* = **Null**

=	Égal à	*expression1* = *expression2*	*expression1* <> *expression2*	*expression1* ou *expression2* = **Null**
<>	Différent de	*expression1* <> *expression2*	*expression1* = *expression2*	*expression1* ou *expression2* = **Null**

Lorsque vous comparez deux expressions, il n'est pas toujours facile de déterminer si elles ont été comparées comme nombres ou comme chaînes.

Le tableau suivant décrit la manière dont les expressions sont comparées, ou ce qui résulte de la comparaison, en fonction du sous-type sous-jacent :

Si	**alors**
Les deux expressions sont numériques	Effectue une comparaison numérique.
Les deux expressions sont des chaînes	Effectue une comparaison de chaînes.
Une expression est numérique et l'autre est une chaîne	L'expression numérique est inférieure à l'expression de chaîne.
Une expression est **Empty** et l'autre est numérique	Effectue une comparaison numérique, en utilisant 0 comme expression **Empty**.
Une expression est **Empty** et l'autre est une chaîne	Effectue une comparaison de chaîne, en utilisant une chaîne de longueur nulle comme l'expression **Empty**.
Les deux expressions sont **Empty**	Les expressions sont égales.

Opérateurs de comparaison

Voir aussi

Is, opérateur
Opérateur, priorité
Opérateur, résumé

Microsoft® Visual Basic® Scripting Edition

Référence du langage

Résumé des opérateurs

Voir aussi

Opérateurs arithmétiques

Opérateurs utilisés pour effectuer des calculs mathématiques.

Opérateurs de comparaison

Opérateurs utilisés pour effectuer des comparaisons.

Opérateurs de concaténation

Opérateurs utilisés pour combiner des chaînes.

Opérateurs logiques

Opérateurs utilisés pour effectuer des opérations logiques.

Microsoft® Visual Basic® Scripting Edition

Résumé des opérateurs

Voir aussi

Priorité des opérateurs

Opérateurs logiques

Microsoft® Visual Basic® Scripting Edition

Référence du langage

And, opérateur

Voir aussi

Description

Utilisé pour effectuer une conjonction logique sur deux expressions.

Syntaxe

result = *expression1* And *expression2*

La syntaxe de l'opérateur And comprend les éléments suivants :

Élément	Description
result	Toute variable numérique.
expression1	Toute expression.
expression2	Toute expression.

Notes

Si, et seulement si, les deux expressions produisent la valeur True, *result* est True. Si l'une ou l'autre produit la valeur False, *result* est False. Le tableau suivant illustre la manière dont *result* est déterminé:

Si *expression1* est	et *expression2* est	*result* est
True	**True**	**True**
True	**False**	**False**
True	**Null**	**Null**
False	**True**	**False**
False	**False**	**False**
False	**Null**	**False**
Null	**True**	**Null**
Null	**False**	**False**
Null	**Null**	**Null**

L'opérateur And effectue aussi une comparaison binaire des bits de position identique dans deux expressions numériques et définit le bit correspondant dans *result* d'après la table de vérité suivante:

Si le bit dans *expression1* est	et le bit dans *expression2* est	*result* est

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

Microsoft® Visual Basic® Scripting Edition

And, opérateur

Voir aussi

Opérateurs logiques
Not, opérateur
Priorité des opérateurs
Résumé des opérateurs
Or, opérateur
Xor, opérateur

Not, opérateur

Voir aussi

Description

Utilisé pour effectuer une négation logique sur une expression.

Syntaxe

result = Not *expression*

La syntaxe de l'opérateur Not comprend les éléments suivants :

Élément	Description
result	Toute variable numérique.
expression	Toute expression.

Notes

Le tableau suivant illustre la manière dont l'élément *result* est déterminé:

Si *expression* est	alors *result* vaut
True	**False**
False	**True**
Null	**Null**

De plus, l'opérateur Not inverse les valeurs de bit de toute variable et définit le bit correspondant dans *result* d'après la table suivante :

Bit dans *expression*	Bit dans *result*
0	1
1	0

Microsoft® Visual Basic® Scripting Edition

Not, opérateur

Voir aussi

And, opérateur
Opérateurs logiques
Priorité des opérateurs
Résumé des opérateurs
Or, opérateur
Xor, opérateur

Microsoft® Visual Basic® Scripting Edition

Or, opérateur

Voir aussi

Description

Utilisé pour effectuer une disjonction logique sur deux expressions.

Syntaxe

result = *expression1* Or *expression2*

La syntaxe de l'opérateur Or comprend les éléments suivants :

Élément	Description
result	Toute variable numérique.
expression1	Toute expression.
expression2	Toute expression.

Notes

Si une ou les deux expressions produisent la valeur True, *result* vaut True. Le tableau suivant illustre la manière dont *result* est déterminé :

Si *expression1* est	et *expression2* est	alors *result* vaut
True	**True**	**True**
True	**False**	**True**
True	**Null**	**True**
False	**True**	**True**
False	**False**	**False**
False	**Null**	**Null**
Null	**True**	**True**
Null	**False**	**Null**
Null	**Null**	**Null**

L'opérateur Or effectue aussi une comparaison binaire des bits de position identique dans deux expressions numériques et définit le bit correspondant dans *result* d'après la table suivante :

Si le bit *expression1* est	et le bit dans *expression2* est	alors *result* vaut

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

Microsoft® Visual Basic® Scripting Edition

Référence du langage

Or, opérateur

Voir aussi

And, opérateur
Opérateurs logiques
Not, opérateur
Priorité des opérateurs
Résumé des opérateurs
Xor, opérateur

Microsoft® Visual Basic® Scripting Edition

Xor, opérateur

Voir aussi

Description

Utilisé pour effectuer une exclusion logique sur deux expressions.

Syntaxe

result = *expression1* Xor *expression2*

La syntaxe de l'opérateur Xor comprend les éléments suivants :

Élément	Description
result	Toute variable numérique.
expression1	Toute expression.
expression2	Toute expression.

Notes

Si une, et une seule, de ces expressions produit la valeur True, *result* vaut True. Toutefois, si l'une ou l'autre expression est Null, *result* vaut également Null. Si aucune expression n'a la valeur Null, l'élément *result* est déterminé conformément au tableau suivant :

Si *expression1* est	et *expression2* est	alors *result* vaut
True	**True**	**False**
True	**False**	**True**
False	**True**	**True**
False	**False**	**False**

L'opérateur Xor effectue aussi une comparaison binaire des bits de position identique dans deux expressions numériques et définit le bit correspondant dans *result* d'après la table suivante :

Si le bit dans *expression1* est	et le bit dans *expression2* est	alors *result* vaut
0	0	0
0	1	1
1	0	1
1	1	0

Microsoft® Visual Basic® Scripting Edition

Xor, opérateur

Voir aussi

And, opérateur
Opérateurs logiques
Not, opérateur
Priorité des opérateurs
Résumé des opérateurs
Or, opérateur

Microsoft® Visual Basic® Scripting Edition

*, opérateur

Description

Utilisé pour multiplier deux nombres.

Syntaxe

*result = number1*number2*

La syntaxe de l'opérateur * comprend les éléments suivants :

Élément	Description
result	Toute variable numérique.
number1	Toute expression numérique.
number2	Toute expression numérique.

Note

Si une ou les deux expressions sont des expressions Null, *result* est Null. Si une expression est Empty, elle est traitée comme s'il s'agissait de 0.

*, opérateur

Voir aussi

\, opérateur
Opérateurs arithmétiques
Priorité des opérateurs
Résumé des opérateurs

Microsoft® Visual Basic® Scripting Edition

\, opérateur

Voir aussi

Description

Utilisé pour diviser deux nombres et retourner un résultat en nombre entier.

Syntaxe

result = number1\number2

La syntaxe de l'opérateur \ comprend les éléments suivants :

Élément	Description
result	Toute variable numérique.
number1	Toute expression numérique.
number2	Toute expression numérique.

Notes

Avant d'effectuer la division, les expressions numériques sont arrondies en expression de sous-type Byte, Integer ou Long.

Si une expression est Null, *result* est également Null. Toute expression qui est Empty est traitée comme 0.

\, opérateur

Voir aussi

*, opérateur
/, opérateur
Opérateurs arithmétiques
Priorité des opérateurs
Résumé des opérateurs

Microsoft® Visual Basic® Scripting Edition

Référence du langage

/, opérateur

Voir aussi

Description

Utilisé pour diviser deux nombres et retourner un résultat en virgule flottante.

Syntaxe

result = *number1* / *number2*

La syntaxe de l'opérateur / comportent les éléments suivants :

Élément	Description
result	Toute variable numérique.
number1	Toute expression numérique.
number2	Toute expression numérique.

Note

Si une ou les deux expressions sont des expressions Null, *result* est Null. Toute expression Empty est traitée comme 0.

/, opérateur

Voir aussi

*, opérateur
\, opérateur
Opérateurs arithmétiques
Priorité des opérateurs
Résumé des opérateurs

Microsoft® Visual Basic® Scripting Edition

Mod, opérateur

Voir aussi

Description

Utilisé pour diviser deux nombres et ne retourner que le reste.

Syntaxe

result = *number1* Mod *number2*

La syntaxe de l'opérateur Mod comprend les éléments suivants :

Élément	Description
result	Toute variable numérique.
number1	Toute expression numérique.
number2	Toute expression numérique.

Notes

L'opérateur modulo, ou reste, divise *number1* par *number2* (en arrondissant en entiers les nombres en virgules flottantes) et ne retourne que le reste comme *result*. Par exemple, dans l'expression suivante, A (qui est l'élément *result*) est égal à 5.

```
A = 19 Mod 6.7
```

Si toute expression est Null, *result* est aussi Null. Toute expression qui est Empty est traitée comme 0.

Microsoft® Visual Basic® Scripting Edition

Mod, opérateur

Voir aussi

Opérateurs arithmétiques
Priorité des opérateurs
Résumé des opérateurs

Une première page simple

Microsoft® Visual Basic® Scripting Edition

Const, instruction

Voir aussi

Description

Déclare des constantes destinées à remplacer des valeurs littérales.

Syntaxe

[Public | Private] Const *constname* = *expression*

La syntaxe de l'instruction Const comprend les éléments suivants:

Élément	Description
Public	Facultatif. Mot clé utilisé au niveau script pour déclarer des constantes accessibles dans toutes les procédures de tous les scripts. Interdit dans les procédures.
Private	Facultatif. Mot clé utilisé au niveau script pour déclarer des constantes accessibles uniquement dans le script où la déclaration est effectuée. Interdit dans les procédures.
constname	Nom de la constante; respecte les conventions standard d'attribution de nom de variable.
expression	Littéral ou autre constante, ou toute combinaison incluant tous les opérateurs arithmétiques ou logiques, à l'exception de **Is**.

Notes

Les constantes sont publiques par défaut. À l'intérieur des procédures, elles sont toujours privées et leur visibilité ne peut pas être modifiée. Dans un script, la visibilité par défaut d'une constante de niveau script peut être modifiée à l'aide du mot clé Private.

Pour combiner plusieurs déclarations de constante sur la même ligne, séparez chaque affectation de constante par une virgule. Lorsque des déclarations de constante sont combinées de cette manière, l'emploi éventuel d'un mot clé Public ou Private s'applique à toutes ces déclarations.

Vous ne pouvez pas utiliser des variables, des fonctions définies par l'utilisateur ou des fonctions VBScript intrinsèques (telles que Chr) dans des déclarations de constante. Par définition, elles ne peuvent pas être des constantes. Vous ne pouvez pas non plus créer de constantes à partir d'une expression impliquant un opérateur, c'est-à-dire que seules les constantes simples sont autorisées. Les constantes déclarées dans une procédure Sub ou Function sont locales à cette procédure. Une constante déclarée à l'extérieur d'une procédure est définie pour l'ensemble du script dans lequel elle est déclarée. Vous pouvez utiliser des constantes à tout endroit où vous pouvez employer une expression.

Remarque Les constantes peuvent documenter automatiquement vos scripts et en simplifier la modification. Contrairement aux variables, les constantes ne peuvent pas être modifiées accidentellement pendant l'exécution de votre script.

Microsoft® Visual Basic® Scripting Edition

Référence du langage

Const, instruction

Voir aussi

Dim, instruction
Function, instruction
Private, instruction
Public, instruction
Sub, instruction

Microsoft® Visual Basic® Scripting Edition

MsgBox, fonction

Description

Affiche un message dans une boîte de dialogue, attend que l'utilisateur clique sur un bouton et retourne une valeur indiquant le bouton choisi par l'utilisateur.

Syntaxe

MsgBox(*prompt*[, *buttons*][, *title*][, *helpfile*, *context*])

La syntaxe de la fonction MsgBox comprend les arguments suivants:

Élément	Description
prompt	Expression de chaîne affichée sous forme de message dans la boîte de dialogue. La longueur maximum de *prompt* est de 1024 caractères environ, selon la largeur des caractères utilisés. Si l'argument *prompt* comporte plusieurs lignes, vous pouvez les séparer par un caractère de retour chariot (**Chr(**13**)**), un caractère de retour à la ligne (**Chr(**10**)**) ou par une combinaison des deux (**Chr(**13**) & Chr(**10**)**) entre chaque ligne.
buttons	Expression numérique correspondant à la somme des valeurs spécifiant le nombre et le type de boutons à afficher, le style d'icône à utiliser, l'identité du bouton par défaut et la modalité du message. Pour les valeurs, reportez-vous à la section ci-après. Si elle est omise, la valeur par défaut de l'argument *buttons* est 0.
title	Expression de chaîne affichée dans la barre de titre de la boîte de dialogue. Si vous omettez l'argument *title*, le nom de l'application s'affiche dans la barre de titre.
helpfile	Expression de chaîne qui identifie le fichier d'aide à utiliser pour fournir l'aide contextuelle de la boîte de dialogue. Si l'argument *helpfile* est fourni, l'argument *context* doit aussi l'être. Non disponible aur les plates-formes 16 bits
context	Expression numérique correspondant au numéro de contexte d'aide affecté par l'auteur de l'Aide à la rubrique d'aide appropriée. Si l'argument *context* est fourni, l'argument *helpfile* doit aussi l'être.

Valeurs

L'argument *buttons* peut prendre les valeurs suivantes:

Constante	Valeur	Description
vbOKOnly	0	Affiche uniquement le bouton OK.
vbOKCancel	1	Affiche les boutons OK et Annuler.
vbAbortRetryIgnore	2	Affiche les boutons Abandon, Réessayer et Ignorer.
vbYesNoCancel	3	Affiche les boutons Oui, Non et Annuler.

vbYesNo	4	Affiche les boutons Oui et Non.
vbRetryCancel	5	Affiche les boutons Réessayer et Annuler.
vbCritical	16	Affiche l'icône Message critique.
vbQuestion	32	Affiche l'icône Demande d'avertissement.
vbExclamation	48	Affiche l'icône Message d'avertissement.
vbInformation	64	Affiche l'icône Message d'information.
vbDefaultButton1	0	Le premier bouton est le bouton par défaut.
vbDefaultButton2	256	Le deuxième bouton est le bouton par défaut.
vbDefaultButton3	512	Le troisième bouton est le bouton par défaut.
vbDefaultButton4	768	Le quatrième bouton est le bouton par défaut.
vbApplicationModal	0	Application modale; l'utilisateur doit répondre au message avant de continuer à travailler dans l'application courante.
vbSystemModal	4096	Système modal; toutes les applications sont suspendues jusqu'à ce que l'utilisateur réponde au message.

Le premier groupe de valeurs (0 à 5) décrit le nombre et le type de boutons affichés dans la boîte de dialogue; le deuxième groupe (16, 32, 48, 64) décrit le style d'icône; le troisième groupe (0, 256, 512, 768) détermine le bouton par défaut; et le quatrième groupe (0, 4096) détermine la modalité du message. Au moment de l'ajout de nombres en vue de créer une valeur finale pour l'argument *buttons*, n'utilisez qu'un seul nombre de chaque groupe.

Valeurs retournées

La fonction MsgBox retourne les valeurs suivantes:

Constante	Valeur	Bouton choisi
vbOK	1	OK
vbCancel	2	Annuler
vbAbort	3	Abandon
vbRetry	4	Réessayer
vbIgnore	5	Ignorer
vbYes	6	Oui
vbNo	7	Non

Notes

Quand les arguments *helpfile* et *context* sont tous deux fournis, l'utilisateur peut

appuyer sur F1 pour afficher la rubrique d'aide correspondant au contexte.

Si la boîte de dialogue affiche un bouton Annuler, le fait d'appuyer sur la touche ÉCHAP a le même effet que de cliquer sur Annuler. Si la boîte de dialogue contient un bouton Aide, l'aide contextuelle est disponible pour la boîte de dialogue. Toutefois, aucune valeur n'est retournée avant qu'un des autres boutons ne soit sélectionné.

Microsoft® Visual Basic® Scripting Edition

Référence du langage

MsgBox, fonction

Voir aussi

InputBox, fonction

Microsoft® Visual Basic® Scripting Edition

InputBox, fonction

Description

Affiche une invite dans une boîte de dialogue, attend que l'utilisateur entre du texte ou choisisse un bouton et retourne le contenu de la zone de texte.

Syntaxe

InputBox(*prompt*[, *title*][, *default*][, *xpos*][, *ypos*][, *helpfile*, *context*])

La syntaxe de la fonction InputBox comprend les éléments suivants :

Élément	Description
prompt	Expression de chaîne qui est affichée sous la forme d'un message dans la boîte de dialogue. La longueur maximum de l'argument *prompt* est environ 1024 caractères, selon la largeur des caractères utilisés. Si l'argument *prompt* se compose de plusieurs lignes, vous pouvez les séparer en utilisant un caractère de retour chariot (**Chr(**13**)**), un caractère de retour à la ligne (**Chr(**10**)**) ou une combinaison de ces deux caractères (**Chr(**13**) & Chr(**10**)**) entre chaque ligne.
title	Expression de chaîne qui est affichée dans la barre de titre de la boîte de dialogue. Si vous omettez l'argument *title*, le nom de l'application s'affiche dans la barre de titre.
default	Expression de chaîne qui est affichée dans la zone de texte comme la réponse par défaut si aucune autre entrée n'est fournie. Si vous omettez l'argument *default*, la zone de texte s'affiche vide.
xpos	Expression numérique qui spécifie, en twips, la distance horizontale entre le bord gauche de la boîte de dialogue et le bord gauche de l'écran. Si l'argument *xpos* est omis, la boîte de dialogue est centrée horizontalement.
ypos	Expression numérique qui spécifie, en twips, la distance verticale entre le bord supérieur de la boîte de dialogue et le haut de l'écran. Si l'argument *ypos* est omis, la boîte de dialogue est positionnée verticalement, de manière approximative, à une distance d'un tiers de la taille de l'écran à partir du haut.
helpfile	Expression de chaîne qui identifie le fichier d'aide à utiliser pour fournir l'aide contextuelle de la boîte de dialogue. Si l'argument *helpfile* est fourni, l'argument *context* doit l'être aussi.
context	Expression numérique qui identifie le numéro de contexte de l'aide affecté par l'auteur de l'Aide à la rubrique d'aide correspondante. Si l'argument *context* est fourni, l'argument *helpfile* doit l'être aussi.

Notes

Lorsque les arguments *helpfile* et *context* sont tous deux fournis, un bouton d'aide est ajouté automatiquement à la boîte de dialogue.

Si l'utilisateur clique sur OK ou appuie sur ENTRÉE, la fonction InputBox retourne

ce qui se trouve dans la zone de texte. Si l'utilisateur clique sur Annuler, la fonction retourne une chaîne de longueur nulle ("").

Microsoft® Visual Basic® Scripting Edition

Référence du langage

InputBox, fonction

Voir aussi

MsgBox, fonction

Do...Loop, instruction

Voir aussi

Description

Répète un bloc d'instructions tant qu'une condition est True ou jusqu'à ce qu'une condition devienne True.

Syntaxe

```
Do [{While | Until} condition]
   [statements]
   [Exit Do]
   [statements]
Loop
```

Vous pouvez aussi utiliser la syntaxe suivante:

```
Do
   [statements]
   [Exit Do]
   [statements]
Loop [{While | Until} condition]
```

La syntaxe de l'instruction Do...Loop comprend les éléments suivants:

Élément	Description
condition	Expression numérique ou expression de chaîne qui est **True** ou **False**. Si *condition* est **Null**, l'élément *condition* est traité comme **False**.
statements	Une ou plusieurs instructions qui sont répétées tant ou jusqu'à ce que l'élément *condition* soit **True**.

Notes

L'instruction Exit Do ne peut être utilisée que dans une structure de contrôle Do...Loop afin de proposer un solution alternative pour quitter une instruction Do...Loop. Vous pouvez placer autant d'instructions Exit Do que vous voulez n'importe où dans l'instruction Do...Loop. Souvent utilisée avec l'évaluation d'une condition (par exemple, l'instruction If...Then), l'instruction Exit Do transfère le contrôle à l'instruction immédiatement après Loop.

Quand elle est utilisée à l'intérieur d'instructions Do...Loop imbriquées, l'instruction Exit Do transfère le contrôle à la boucle située au niveau d'imbrication supérieur à la boucle dans laquelle elle se déroule.

Microsoft® Visual Basic® Scripting Edition

Référence du langage

Do...Loop, instruction

Voir aussi

Exit, instruction
For...Next, instruction
While...Wend, instruction

Microsoft® Visual Basic® Scripting Edition

Exit, instruction

Voir aussi

Description

Quitte un bloc du code Do...Loop, For...Next, Function ou Sub.

Syntaxe

Exit Do

Exit For

Exit Function

Exit Sub

La syntaxe de l'instruction Exit prend les formes suivantes :

Instruction	Description
Exit Do	Fournit un moyen de quitter une instruction **Do...Loop**. Elle ne peut être utilisée qu'à l'intérieur d'une instruction **Do...Loop**. **Exit Do** transfère le contrôle à l'instruction suivant l'instruction **Loop**. Quand elle est utilisée dans des instructions **Do...Loop** imbriquées, l'instruction **Exit Do** transfère le contrôle à la boucle située au niveau d'imbrication supérieur à la boucle dans laquelle elle se produit.
Exit For	Fournit un moyen de quitter une boucle **For**. Elle ne peut être utilisée que dans une boucle **For...Next** ou **For Each...Next**. **Exit For** transfère le contrôle à l'instruction suivant l'instruction **Next**. Quand elle est utilisée dans des boucles **For** imbriquées, l'instruction **Exit For** transfère le contrôle à la boucle située au niveau d'imbrication supérieur à la boucle dans laquelle elle se produit.
Exit Function	Quitte immédiatement la procédure **Function** dans laquelle elle apparaît. L'exécution se poursuit avec l'instruction suivant l'instruction ayant appelé la procédure **Function**.
Exit Sub	Quitte immédiatement la procédure **Sub** dans laquelle elle apparaît. L'exécution se poursuit avec l'instruction suivant l'instruction ayant appelé la procédure **Sub**.

Microsoft® Visual Basic® Scripting Edition

Référence du langage

Exit, instruction

Voir aussi

Do...Loop, instruction
For Each...Next, instruction
For...Next, instruction
Function, instruction
Sub, instruction

Microsoft® Visual Basic® Scripting Edition

For Each...Next, instruction

Voir aussi

Description

Répète un groupe d'instructions pour chaque élément d'un tableau ou d'une collection.

Syntaxe

```
For Each element In group
   [statements]
   [Exit For]
   [statements]
Next [element]
```

La syntaxe de l'instruction For Each...Next comprend les éléments suivants:

Élément	Description
element	Variable employée pour effectuer une itération sur les éléments de la collection ou du tableau. Pour les collections, l'argument *element* peut uniquement être une variable de type **Variant**, une variable de type **Object** générique, ou n'importe quelle variable d'objet Automation spécifique. Pour les tableaux, l'argument *element* peut être uniquement une variable de type **Variant**.
group	Nom d'une collection d'objets ou d'un tableau.
statements	Une ou plusieurs instructions pouvant être exécutées sur chaque élément d'un groupe indiqué par *group*.

Notes

Le bloc d'instruction For Each peut être tapé si l'entité *group* comporte au moins un élément. Une fois la boucle, toutes les instructions de celle-ci sont exécutées pour le premier élément de l'entité *group*. Puis, tant qu'il reste des éléments dans l'entité *group*, les instructions de la boucle continuent à s'exécuter pour chaque élément. Lorsqu'il n'y a plus d'éléments dans l'entité *group*, le programme sort de la boucle et exécute l'instruction se trouvant immédiatement après l'instruction Next.

L'instruction Exit For ne peut être utilisée qu'à l'intérieur d'une structure de contrôle For Each...Next ou For...Next pour fournir un autre mode de sortie. La boucle peut inclure un nombre illimité d'instructions Exit For. L'instruction Exit For est souvent employée avec l'évaluation d'une condition (par exemple, If...Then), et transfère le contrôle à l'instruction se trouvant immédiatement après Next.

Vous pouvez imbriquer des boucles For Each...Next en plaçant une boucle For Each...Next à l'intérieur d'une autre. Cependant, chaque *element* de boucle doit être unique.

Remarque Si vous omettez l'argument *element* dans une instruction **Next**, l'exécution se poursuit comme si vous l'aviez incluse. Si une instruction **Next** est rencontrée avant son instruction **For** correspondante, une erreur se produit.

Microsoft® Visual Basic® Scripting Edition

Référence du langage

For Each...Next, instruction

Voir aussi

Do...Loop, instruction
Exit, instruction
For...Next, instruction
While...Wend, instruction

Microsoft® Visual Basic® Scripting Edition

For...Next, instruction

Voir aussi

Description

Répète un groupe d'instructions un nombre spécifié de fois.

Syntaxe

```
For counter = start To start [Step increment]
   [statements]
   [Exit For]
   [statements]
Next
```

La syntaxe de l'instruction For...Next comporte les éléments suivants:

Élément	Description
counter	Variable numérique utilisée comme compteur de boucles. La variable ne peut être un élément de tableau ou un élément d'un type défini par l'utilisateur.
start	Valeur initiale de *counter*.
start	Valeur finale de *counter*.
increment	Quantité par laquelle *counter* change à chaque accomplissement de la boucle. Si cet élément n'est pas spécifié, la valeur par défaut de *increment* est 1.
statements	Une ou plusieurs instructions entre **For** et **Next** qui sont exécutées le nombre de fois spécifié.

Notes

L'argument *increment* peut être positif ou négatif. La valeur de l'argument *increment* détermine le traitement de la boucle de la manière suivante :

Valeur	La boucle s'exécute si
Positif ou 0	*counter* <= *start*
Négatif	*counter* >= *start*

Une fois que la boucle démarre et que toutes les instructions sont exécutées, l'argument *increment* est ajouté à *counter*. À ce point, les instructions contenues dans la boucle sont à nouveau exécutées (sur la base du même test ayant provoqué l'exécution initiale de la boucle) ou la boucle est quittée et l'exécution se poursuit avec l'instruction suivant l'instruction Next.

Conseil Changer la valeur de *counter* quand une boucle est en cours d'exécution rendra la lecture et le débogage de votre code plus difficile.

L'instruction Exit For ne peut être utilisée que dans une structure de contrôle For Each...Next ou For...Next pour fournir un autre moyen de quitter. Vous pouvez placer autant d'instructions Exit For que vous voulez n'importe où dans la boucle. L'instruction Exit For est souvent utilisée avec l'évaluation d'une condition (par exemple, If...Then) et transfère le contrôle à l'instruction succédant immédiatement à Next.

Vous pouvez imbriquer For...Next en plaçant une boucle For...Next dans une autre. Donnez à chaque boucle un nom de variable unique comme son élément *counter*. La construction suivante est correcte :

```
For I = 1 To 10
    For J = 1 To 10
        For K = 1 To 10
        . . .
        Next
    Next
Next
```

Microsoft® Visual Basic® Scripting Edition

Référence du langage

For...Next, instruction

Voir aussi

Do...Loop, instruction
Exit, instruction
For Each...Next, instruction
While...Wend, instruction

While...Wend, instruction

Voir aussi

Description

Exécute une série d'instructions tant qu'une condition donnée est True.

Syntaxe

```
While condition
   [statements]
Wend
```

La syntaxe de l'instruction While...Wend comprend les éléments suivants :

Élément	Description
condition	Expression numérique ou expression de chaîne qui produit la valeur **True** ou **False**. Si *condition* est **Null**, l'élément *condition* est traité comme étant **False**.
statements	Une ou plusieurs instructions exécutées alors que condition est **True**.

Notes

Si *condition* est True, toutes les instructions contenues dans *statements* sont exécutées jusqu'à ce que l'instruction Wend soit rencontrée. L'instruction While prend ensuite le contrôle et l'élément *condition* est à nouveau vérifié. Si *condition* est toujours True, le processus est répété. Si condition n'est pas True, l'exécution reprend en commençant par l'instruction succédant à l'instruction Wend.

Des boucles While...Wend peuvent être imbriquées à tous les niveaux. Chaque instruction Wend correspond à l'instruction While la plus récente.

Conseil L'instruction **Do...Loop** fournit un moyen plus structuré et plus souple d'effectuer une itération en boucle.

Microsoft® Visual Basic® Scripting Edition

Référence du langage

While...Wend, instruction

Voir aussi

Do...Loop, instruction

Microsoft® Visual Basic® Scripting Edition

If...Then...Else, instruction

Description

Exécute conditionnellement un groupe d'instructions, en fonction de la valeur d'une expression.

Syntaxe

If *condition* Then *statements* [Else *elsestatements*]

Ou, vous pouvez utiliser la syntaxe suivante, plus polyvalente:

```
If condition Then
   [statements]
[ElseIf condition-n Then
   [elseifstatements]] . . .
[Else
   [elsestatements]]
End If
```

La syntaxe de l'instruction If...Then...Else comporte les éléments suivants:

Élément	Description
condition	Un ou plusieurs types d'expressions suivants: Une expression numérique ou une expression de chaîne qui produit la valeur **True** ou **False**. Si *condition* est **Null**, l'élément *condition* est traité comme **False**. Une expression de la forme **TypeOf** *objectname* **Is** *objecttype*. L'élément *objectname* est toute référence d'objet et l'élément *objecttype* est tout type d'objet valide. L'expression est **True** si *objectname* est le type d'objet spécifié par *typeobjet*; dans le cas contraire, sa valeur est **False**.
statements	Une ou plusieurs instructions séparées par deux-points; exécutées si *condition* est **True**.
condition-n	*Condition* identique.
elseifstatements	Une ou plusieurs instructions exécutées si la *condition-n* est **True**.
elsestatements	Une ou plusieurs instructions exécutées si aucune expression de *condition* ou *condition-n* n'est **True**.

Notes

Vous pouvez utiliser la forme en ligne simple (première syntaxe) pour les tests courts et simples. Toutefois, la forme en bloc (seconde syntaxe) fournit une structure plus solide et une plus grande souplesse que la forme en ligne simple, et elle est souvent plus facile à lire, à mettre à jour et à déboguer.

Remarque Avec la syntaxe en ligne simple, il est possible de provoquer l'exécution de plusieurs instructions comme résultat d'une décision **If...Then**, mais elles doivent toutes être sur la même ligne et séparées par deux-points, comme dans l'instruction suivante:

```
If A > 10 Then A = A + 1 : B = B + A : C = C + B
```

Lors de l'exécution d'un bloc If (seconde syntaxe), *condition* est testé. Si *condition* est True, les instructions suivant Then sont exécutées. Si *condition* est False, chaque clause ElseIf (s'il en existe) est évaluée à tour de rôle. Quand une condition True est trouvée, les instructions suivant l'élément Then sont exécutées. Si aucune des instructions ElseIf n'est True (ou s'il n'existe pas de clause ElseIf), les instructions suivant Else sont exécutées. Après l'exécution des instructions suivant Then ou Else, l'exécution se poursuit avec l'instruction suivant End If.

Les clauses Else et ElseIf sont toutes deux facultatives. Vous pouvez intégrer autant d'instructions ElseIf que vous voulez dans un bloc If, mais aucune ne peut apparaître après la clause Else. Les instructions de blocs If peuvent être imbriquées, autrement dit, se contenir l'une l'autre.

Ce qui suit le mot clé Then est examiné pour déterminer si une instruction est un bloc If ou non. Si tout élément autre qu'un commentaire apparaît après Then sur la même ligne, l'instruction est traitée comme une instruction If en ligne simple.

Une instruction contenant des blocs If doit être la première instruction sur une ligne. Le bloc If doit terminer une instruction End If.

Microsoft® Visual Basic® Scripting Edition

Select Case, instruction

Voir aussi

Description

Exécute un groupe d'instructions parmi plusieurs, en fonction de la valeur d'une expression.

Syntaxe

```
Select Case testexpression
   [Case expressionlist-n
      [statement-n]] . . .
   [Case Else expressionlist-n
      [elsestatements-n]]
End Select
```

La syntaxe de l'instruction Select Case comporte les éléments suivants :

Élément	Description
testexpression	Toute expression numérique ou de chaîne.
expressionlist-n	Requis si **Case** apparaît. Liste délimitée d'une ou de plusieurs expressions.
statements-n	Une ou plusieurs instructions exécutées si *testexpression* correspond à un élément de *expressionlist-n*.
elsestatements	Une ou plusieurs instructions exécutées si *testexpression* ne correspond à aucune des clauses **Case**.

Notes

Si *testexpression* correspond à une expression Case *expressionlist*, les instructions suivant cette clause Case sont exécutées jusqu'à la clause Case suivante ou, pour la dernière clause, jusqu'à End Select. L'instruction suivant End Select prend ensuite le contrôle. Si *testexpression* correspond à une expression *expressionlist* dans plusieurs clauses Case, seules les instructions suivant la première correspondance sont exécutées.

La clause Case Else est utilisée pour indiquer les *elsestatements* à exécuter si aucune correspondance n'était trouvée entre l'expression *testexpression* et une *expressionlist* dans toutes les autres sélections Case. Bien que ce ne soit pas obligatoire, il est judicieux d'insérer une instruction Case Else dans votre bloc Select Case pour gérer les valeurs *testexpression* imprévues. Si aucune Case *expressionlist* ne correspond à *testexpression* et s'il n'y a pas d'instruction Case Else, l'exécution continue à partir de l'instruction suivant End Select.

Les instructions Select Case peuvent être imbriquées. Chaque instruction Select Case imbriquée doit avoir une instruction End Select correspondante.

Microsoft® Visual Basic® Scripting Edition

Référence du langage

Select Case, instruction

Voir aussi

If ... Then ... Else, instruction

Microsoft® Visual Basic® Scripting Edition

Eqv, opérateur

Voir aussi

Description

Utilisé pour effectuer une équivalence logique sur deux expressions.

Syntaxe

result = *expression1* Eqv *expression2*

La syntaxe de l'opérateur Eqv comprend les éléments suivants :

Élément	Description
result	Toute variable numérique.
expression1	Toute expression.
expression2	Toute expression.

Notes

Si l'une ou l'autre expression est Null, *result* est Null également. Si aucune des expressions n'est Null, *result* est déterminé en fonction du tableau suivant :

Si *expression1* est	et *expression2* est	*result* est
True	**True**	**True**
True	**False**	**False**
False	**True**	**False**
False	**False**	**True**

L'opérateur Eqv effectue une comparaison binaire des bits ayant une position identique dans deux expressions numériques, et définit le bit correspondant dans *result* d'après la table de vérité suivante :

Si le bit dans *expression1* est	et si le bit dans *expression2* est	*result* est
0	0	1
0	1	0
1	0	0
1	1	1

Microsoft® Visual Basic® Scripting Edition

Référence du langage

Eqv, opérateur

Voir aussi

Imp, opérateur
Opérateurs logiques
Priorité des opérateurs
Résumé des opérateurs

Microsoft® Visual Basic® Scripting Edition

Imp, opérateur

Voir aussi

Description

Utilisé pour effectuer une implication logique sur deux expressions.

Syntaxe

result = *expression1* Imp *expression2*

La syntaxe de l'opérateur Imp comprend les éléments suivants :

Élément	Description
result	Toute variable numérique.
expression1	Toute expression.
expression2	Toute expression.

Notes

Le tableau suivant illustre la manière dont est déterminé l'élément *result* :

Si *expression1* est	et *expression2* est	Alors *result* vaut
True	**True**	**True**
True	**False**	**False**
True	**Null**	**Null**
False	**True**	**True**
False	**False**	**True**
False	**Null**	**True**
Null	**True**	**True**
Null	**False**	**Null**
Null	**Null**	**Null**

L'opérateur Imp effectue une comparaison binaire des bits de position identique dans deux expressions numériques et définit le bit correspondant dans *result* d'après la table suivante :

Si le bit dans *expression1* est	et le bit dans *expression2* est	Alors *result* vaut

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

Microsoft® Visual Basic® Scripting Edition

Référence du langage

Imp, opérateur

Voir aussi

Eqv, opérateur
Opérateurs logiques
Priorité des opérateurs
Résumé des opérateurs

Microsoft® Visual Basic® Scripting Edition

Option Explicit, instruction

Description

Utilisée au niveau du script pour forcer la déclaration explicite de toutes les variables contenues dans ce script.

Syntaxe

```
Option Explicit
```

Notes

Si elle est utilisée, l'instruction Option Explicit doit apparaître dans un script avant toute procédure.

Quand vous utilisez l'instruction Option Explicit, vous devez déclarer explicitement toutes les variables en utilisant les instructions Dim, Private, Public ou ReDim. Si vous essayez d'utiliser le nom d'une variable non déclarée, une erreur se produit.

Conseil Utilisez **Option Explicit** pour éviter de taper incorrectement le nom d'une variable existante ou de créer une confusion dans le code si la portée de la variable n'est pas claire.