Guía de referencia LibreOffice

LibreOffice Basic n.º 9
Tipos Estructurados

Intermedio

v. 1.01 – 03/11/2019

Redactado con LibreOffice v. 5.3.3 – Plataforma: Todas

Matrices

Reagrupación de elementos similares, aplicando un índice interno (Long).

Dimensiones

Una matriz puede contener varias dimensiones (max: 60).

Declaración

La base predet. del índice es `0` (cero) Modificable por `Option Base 1` (poco usado).

En la tabla siguiente: `ix` = índice.

Matrices estáticas	*Características predefinidas*
Una dimension («vector»)	
`Dim T(ixMin To ixMax) As UnTipo`	`Dim T(1 To 12) As UnTipo` 1 dimensión, 12 elementos, (ix = 1)
O bien `Dim T(ixMax) As UnTipo`	`Dim T(9)`: 1 dimensión, 10 elementos, (ix = 0) (se empieza a contar desde cero)
Varias dimensiones	
`Dim T(ixMin1 To ixMax1, ixMin2 To ixMax2) As UnTipo`	Varias dimensiones (en el ejemplo 2). `Dim T(1 To 12, 1 To 31) As Integer` 2 dimensiones, 12 y 31 elementos (ix = 1)
O bien `Dim T(ixMax1, ixMax2, ...) As UnTipo`	`Dim T(2,4) As String`: 2 dimensiones. 3 elementos para la 1ª, 5 para la 2ª.(ix = 0)

Matrices dinámicas	*Las características se definen durante ejec.*
`Dim Array1() As UnTipo` `Dim Array1 As Variant`	Matriz de dimensiones desconocidas. Después será necesario usar `ReDim`

☞ Declarada de tipo `Variant`, una matriz puede contener elementos de distintos tipos.

Matrices anidadas (Nested arrays/Jagged matrices)

O matrices de matrices. Ej: usadas para acceder a los datos de intervalos en Calc.

Una matriz externa (de variants) tiene por elementos matrices de datos:

```
Dim T As Variant
T = Array(Array(1, 2, 3), Array(10, 20, 30), Array(7, 8, 9))
```

`T(0)(0)` vale `1`; `T(2)(2)` vale `9`, etc.

Acceso a un elemento de la matriz

Por índice :

```
Dim Matriz(9) As Integer            Dim Matriz(11, 31) As Integer
Matriz(5) = 123 '(modifica el 5º elem).  Matriz(5, 28) = 123
```

Funciones e instrucciones relativas a la matriz

`Option Base 1`	(Instrucción al inicio de módulo – sólo se aplica al módulo actual) Fuerza a los índices de las matrices a empezar por 1 en lugar de 0.
`IsArray()`	Devuelve `True` si la variable es de tipo matriz (array). `OK = IsArray(Matriz) '(OK obtiene el valor true)`
`Array()`	Devuelve el array creado con valores variados `Matriz = Array("Uno", 2, Now()) 'matriz de variants`
`Redim`	(instrucción) Redimensiona una matriz Con pérdida de datos: `Redim Matriz(dimension)` Sin pérdida de datos: `Redim Preserve Matriz(dimension)`
`Erase`	(Instrucción) Borra el contenido de la matriz `Erase Matriz` Si la matriz es dinámica la libera de la memoria.
`LBound()`	Devuelve el límite inferior si no se indica es el índice de la primera dimensión, o indicando la dimensión: `LBound(Matriz, 2)`
`UBound()`	Devuelve el límite superior (igual que en el caso anterior). ☞ La matriz no tendrá dimensiones definidas si `UBound(Matriz) = -1` y `LBound(Matriz) = 0`
`Split()`	Crea una matriz(vector) cortando una cadena mediante delimitador. `T=Split("C:/file.txt", "/") → T(0)="C:", T(1)="file.txt"`
`Join()`	Fusiona los elementos de una matriz (vector) para obtener una cadena. `Join(T, "|") → "C:|file.txt" (aquí se usa un delimitador)`

Verificación de una matriz

Una variable «Matriz» sólo se procesa si pasa los siguientes (3) test

1. ¿Existe la variable?	`Not IsNull(Matriz)`
2. ¿es una matriz?	`IsArray(Matriz)`
3. ¿Están definidas sus dimensiones?	`UBound(Matriz) >= LBound(Matriz)`

Recorrer una matriz de una dimensión (vector)

Por índices

```
Dim Matriz(9) As Integer, i As Long
For i = LBound(Matriz) To UBound(Matriz)
  Print Matriz(i)
Next i
```

Por elementos

```
Dim Val As 'tipo compatible con todos los elementos
For Each Val In Matriz
  Print Val
Next
```

Recorrer una matriz de dos dimensiones

```
Dim Matriz(2, 4)   '3 filas, 4 columnas
Dim i As Long, j As Long
For i = LBound(Matriz) To Ubound(Matriz)
  For j = Lbound(Matriz, 2) To UBound(Matriz, 2)
    Print Matriz(i, j)
  Next j
Next i
```

Ordenación de matrices

☞ No existe un método predefinido, (busque `QuickSort o Bubblesort` en la web).

Duplicar matrices

Método General	Un bucle (`For..Next`) que copie los valores entre 2 matrices (el proceso puede ser muy largo)

Truco	Por asignación y después usando `ReDim Preserve`:
	`Matriz2 = Matriz1  'Dos matrices -> mismos valores`
	`ReDim Preserve Matriz2(Tamaño)  '-> 2 matrices distintas`
	☞ Se aplica sólo a matrices de valores simples (no objeto).

Usar una matriz en subrutinas

Como parámetro de una subrutina

Sub con parámetro matriz	`Sub UseArray(ByRef Matriz() As String)`
Llamada a la subrutina de la Sub	`Dim Matriz(9) As String` `UseArray(Matriz)`

Como resultado de una función

Función con resultado matriz	`Function GetArray() As Integer()` `GetArray = MatrizDeEnteros()`
La función devuelve	`Dim Matriz As Integer` `Matriz = GetArray()`

Tablas y rangos de libros

(Vea la *Guía de referencia n.º 3*)

Tipos personalizados (Custom types)

Permiten añadir varios valores (miembros) en un tipo de valor único.

Simplifican la manipulación de datos, paso de parámetros o resultado de función.

Tipo de miembros: sin definir o presonalizado(**excepto** matriz).

Declaración de tipo

```
Type MiTipoPerso
  UnMiembro As TipoSimple
  OtroMiembro As OtroTipoSimple
End Type
```

```
Type Suceso
  Nombre As String
  FechaHora As Date
End Type
```

Declaración de una variable de un tipo personalizado

```
Dim UnaVar As MiTipoPerso
```

Limitacion: un tipo personalizado sólo es visible *en el módulo que se declara*. La declaración `As MiTipo` por tando sólo es posible en el seno del **mismo** módulo donde se ha declarado el tipo `MiTipo`. Vea Función de «fábrica» (Factory) / Accesor (accessor).

Uso

Asignación	`VarPerso.UnMiembro = UnValor`
Lectura	`UnaVar = VarPerso.OtroMiembro`

La palabra clave With
Permite agrupar las referencias a los elementos.
	`With VarPerso`
	`  .UnMiembro = UnValor`
	`End With`
	☞ Observe la presencia del punto.

Colecciones

Estructura que permite el acceso rápido a los datos al indexarlos.

Una colección se manipula como un tipo `Object`.

Los elementos almacenados pueden ser de cualquier tipo, incluido `Object`.

La clave de índice es de tipo `String`. En una colección cada clave es **única**.

Declarar una colección	`Dim oColec As New Collection`
Añadir un elemento	

Con clave	`Ocolec.Add(Elt, "Clave")`
	☞ Acceso posterior por clave o por índice.
	☞ La clave no es sensible a mayúsculas.
	☞ Es posible precisar la ubicación del elemento añadido mediante `Before/After`:
	`oColl.Add(Elt, "Clave", After:="Clave0")`

Sin clave	`oColl.Add(Elt)` ☞ Acceso posterior sólo por índice.
Accceder a un elemento	
Por su clave	`Valor = oColl("Clave")`
Por su índice	`Valor = oColl.Item(Index)`
Remplazar un elemento	Misma clave, nuevo valor de elemento. ☞ **Eliminar** y después **Añadir**
Eliminar un elemento	
Por su clave	`oColl.Remove("Clave")`
Por su índice	`oColl.Remove(Index)`
Eliminar todos los elementos	`ReDim oColl As New Collection`
Número de elementos	`Numero = oColl.Count`

Verificación de la existencia de un elemento

Intento de acceso y tratamiento del error eventual (la función sig devuelve true o false)

```
Function ExistsItem(ByRef pColl As Object, pKey As String) As
    Boolean
  Dim Item As Variant, Exists As Boolean
  On Local Error Goto ErrHandler
  Exists = False
  Item = pColl(pKey)  'si pKey no existe -> ErrHandler:
  Exists = True
  ErrHandler:
    'do nothing
  ExistsItem = Exists
End Function
```

Recorrer una colección

Puede obtener los elementos de una colección recorriéndola.

☞ No hay posibilidad de listar las claves.

Por índices

```
For i = 1 To oColl.Count
```

```
  Valor = oColl.Item(i)      'acceso al dato
Next i
```

Por acceso directo a los elementos

```
Dim Element As 'tipo compatible con los elementos de la colección.
For Each Element In oColl
  'Hacer algo con el elemento(dato)
Next
```

Creación de clases en Basic

Embrión de programación orientada a objetos (POO) (OOP en inglés).

☞ Limites: No tiene herencia (utilice la delegation), No tiene polimorfismo

Una clase LibreOffice Basic podría entenderse como un tipo personalizado mejorado, al que se añade un comportamiento (funciones o subrutinas).

Vocabulario

Módulo de clase	**Módulo de código** destinado a contener las declaraciones de la clase
Clase	**Tipo** que permite crear (instanciar) variables objeto.
Sucesos	Métodos para interceptar la **creación** y **destrucción** del objeto.
Miembro	**Variable** interna de una clase.
Propiedad	Refleja el **estado** del objeto.
Método	Realiza una **acción** con el objeto.
Instancia	**Objeto** creado a partir de un tipo clase.

Especificar una clase

Las especificaciones de una clase (miembros, propiedades, métodos) se hacen en el interior de un módulo de código dedicado. En LibreOffice Basic, este módulo no se distingue de un módulo estándar más que por sus opciones iniciales.

☞ Consejo adopte una convención de nombres para los módulos de clase.

Opciones iniciales

Al principio del modulo de clase indique las opciones:

```
Option Explicit  'como habitualmente
Option Compatible
Option ClassModule
```

Variables miembros

Son internas, por tanto declaradas `Private`.

☞ Los miembros de una clase **sólo** deben llamarse a través de propiedades creadas para este propósito **nunca** diréctamente a través de la instancia,.

```
Private mNombre As String
Private mSheet As Object
```

Sucesos (Events)

Son dos subrutinas internas por tanto, declaradas `Private`.

Constructor

```
Private Sub Class_Initialize()
```

Para iniciar el objeto durante su creacción.

Destructor

```
Private Sub Class_Terminate()
```

Para purgar sus componentes internos durante su destrucción.

☠ **Fallo de seguridad**. Se recomienda encarecidamente **no incluir este destructor en sus clases**: debido a un error de implementación en VisualBasic, Class_Terminate () constituye una vulnerabilidad de seguridad y, como tal, es rechazado por los antivirus (consulte el dertificado CVE-2018-8174).

☞ **Limitación**: no es posible pasar parámetros a estas subrutinas.

Propiedades (Properties)

Propiedad = **estado** del objeto.

Son visibles desde el exterior, por tanto declarados `Public`.

De lectura (Get) (todo tipo de datos, incluidos objetos)

```
Public Property Get Name() As String
  Name = mName
End Property
```

De escritura (Let) (todo tipo de datos excepto objetos)

```
Public Property Let Name(ByRef pName As String)
  mName = pName
End Property
```

De escritura (Set) (sólo objetos)

```
Public Property Set Sheet(ByRef pSheet As Object)
  Set mSheet = pSheet
End Property
```

☞ Es posible abandonar prematuarmente una propiedad mediante `Exit Property`

Una clase puede contener propiedades de lectura y escritura: Escriba sucesivamente las dos propiedades `Get` y `Let`/`Set` si es necesario.

Las propiedades pueden acceder a los miembros, propiedades y métodos de la clase.

Métodos (Methods)

Método = **action** sobre/de el objeto.

`Sub` y `Function` propias de la clase, internas (`Private`) o visibles (`Public`). Se escriben como `Sub` o `Function` estándar precedidas de la palabra clave `Public` o `Private`. Estas tienen acceso a los miembros y propiedades de la clase.

Utilización de las clases en Basic

Declarar/crear un objeto

Instanciación inmediata

```
Set MiObjeto = MiClase
```

Instanciación diferida (en la 1ª llamada)

```
Dim MiObjeto As New MiClase
MiObjeto = New MiClase
```

El constructor de la clase se llama en el momento de la instanciación del objeto.

No se puede declarar un objeto `As New MiClase` fuera de la bibliloteca en la que el módulo clase existe. Por lo general deberá declarar el objeto `As Object`.

☠ **Limitación**: Una clase **no** es visible fuera de la biblioteca en la que se declara. Vea: Función de «fábrica» (Factory) / Accesor (accessor).

Acceder a las propiedades y métodos de un objeto

Sintaxis de acceso a los elementos de un objeto:

`objet.propiedad` u `objet.metodo`

Como para los tipos personalicados, puede utilizar la sintaxis `With..End With`

Liberar un objeto

Cuando un objeto no sea ya necesario, puede destruirlo: `Set oObjet = Nothing`

En ese momento se hace una llamada al destructor de la clase

☞ Esta instrucción no es totalmente necesaria en las `Sub` o `Function`, puesto que sus variables internas se destruyen a la salida. Sin embargo, la destrucción efectiva de la variable no está bajo su control.

La instrucción `Set oObjet = Nothing`:
– muestra claramente la intención.
– asegura el control del momento de destrucción del objeto.

Función de «fábrica» (Factory) / Accesor (accessor)

Cuestión de visibilidad de tipos y clases personalizados

- Un **tipo personalizado** sólo es visible en el **módulo** donde se ha declarado.
- Una **clase** sólo es visible en la **biblioteca** donde se ha declarado

Para solucionar este problema, cree una función «fábrica» (o accesor) para crear las variables. A esta función se la podrá llamar desde cualquier módulo o biblioteca.

	☠ Uso	☠ Visibilidad	☞ Fábrica creada en	☞ Declarado
Tipo person.	`As MiTipo`	Mismo módulo	Mismo módulo	`As Variant`
Clase	`As MiClase`	Misma biblioteca	Misma biblioteca, otro módulo	`As Object`

Creacción de la función «fábrica» / accesor

☆ La función fábrica puede también utilizarse para inicializar la variable.

Tipos personalizados

```
Function CreateMiTipoPerso() As MiTipoPerso
  Dim oVar As MiTipoPerso
  CreateMiTipoPerso = oVar
End Function
```

Clases

```
Function CreateMiClase() As MiClase
  Dim oVar As New MiClase
  CreateMiClase = oVar
End Function
```

Uso de la función «fábrica» / accesor

Tipos personalizados

```
Dim MaVar As Variant
MaVar = CreateMiTypoPerso()
```

Clases

```
Dim MaVar As Object
MaVar = CreateMiClase()
```


Creditos

Autor : Jean-François Nifenecker – jean-francois.nifenecker@laposte.net

Somos como enanos sentados sobre los hombros de gigantes. Si vemos más cosas y más lejanas que ellos, no es por la perspicacia de nuestra visión, ni por nuestra grandeza, sino porque son ellos los que nos elevan. (Bernard de Chartres [atribuido])

Historial

Versión	Fecha	Comentarios
1.0	20/04/2019	Primera versión.
1.01	03/11/2019	Correcciones
	15/04/2021	Traducción al español :B. Antonio Fernández

El documento original se puede obtener en la Wiki francesa de publicaciones de L.O.