

# Elementos del lenguaje.

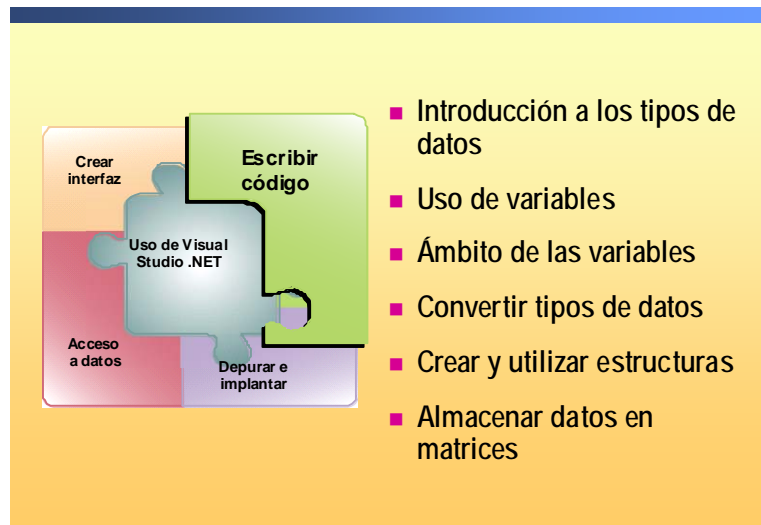
## Variables y estructuras de datos

### Índice

Descripción	1
Lección: Introducción a los tipos de datos	2
Lección: Uso de variables	8
Lección: ámbito de una variable	18
Lección: Convertir tipos de datos	29
Lección: Crear y utilizar estructuras	35
Lección: Almacenar datos en matrices	44



# Descripción



\*\*\*\*\*

## Introducción

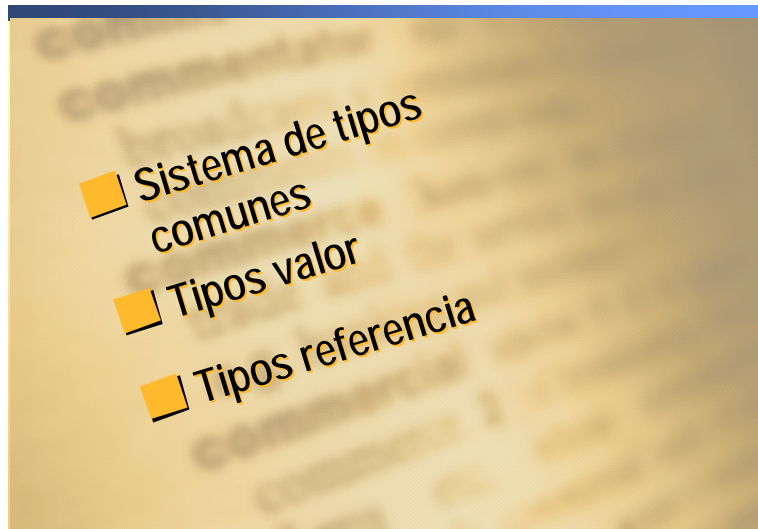
Cuando su aplicación se está ejecutando, utiliza variables para almacenar valores en ubicaciones temporales de memoria, de modo que puede procesar datos y realizar cálculos. Este módulo explica cómo nombrar, declarar, asignar valores y utilizar variables. También describe cómo almacenar datos en una matriz.

## Objetivos

En este módulo, aprenderá a:

- Describir los diversos tipos de datos que puede utilizar para declarar variables.
- Nombrar, declarar, inicializar y utilizar variables y constantes.
- Declarar variables con diferentes niveles de ámbito.
- Crear estructuras de datos definidas por el usuario.
- Convertir valores de variables de un tipo de datos en otro.
- Almacenar datos en matrices.

# Lección: Introducción a los tipos de datos



\*\*\*\*\*

## Introducción

El *tipo de datos* de un elemento de programación hace referencia a la clase de datos que puede contener y a cómo se almacenan los datos. El sistema de tipos comunes (*Common Type System – CTS*) define los tipos de datos que soporta el Common Language Runtime. Visual Basic .NET utiliza tipos de datos que se corresponden directamente con los tipos de datos del sistema de tipos comunes.

Esta lección describe el sistema de tipos comunes, explica cómo Visual Basic .NET implementa los tipos comunes, y cómo escoger los tipos de datos adecuados para una variable determinada.

## Estructura de la lección

Esta lección incluye los siguientes temas y actividades:

- ¿Qué es el sistema de tipos comunes?
- Tipos de datos
- Cómo escoger un tipo de datos
- Práctica: Escoger tipos de datos

## Objetivos de la lección

En esta lección, aprenderá a:

- Describir los tipos disponibles en el sistema de tipos comunes.
- Explicar la diferencia entre las variables de tipo-valor y de tipo-referencia.
- Enumerar algunos de los tipos de datos más utilizados disponibles en Visual Basic .NET.
- Elegir el tipo de datos adecuado para una variable.

## ¿Qué es el sistema de tipos comunes?



\*\*\*\*\*

### Introducción

El sistema de tipos comunes define cómo se declaran, utilizan y gestionan los tipos en el Common Language Runtime. Cada tipo de datos utilizado en Visual Basic .NET corresponde directamente a un tipo definido en el sistema de tipos comunes.

### Ventajas del sistema de tipos comunes

El sistema de tipos comunes tiene una gran importancia en la creación de aplicaciones para la plataforma Microsoft .NET. Hace posible que un desarrollador pueda crear un proyecto en Visual Basic .NET e integrarlo con un componente creado por otro desarrollador en Microsoft Visual C#™ y una función escrita por un tercer desarrollador en otro lenguaje compatible con .NET. Todas estas piezas pueden integrarse en una única solución. Los compiladores y herramientas de Microsoft Visual Studio® .NET y el Common Language Runtime dependen del sistema de tipos comunes para proporcionar:

- Integración entre lenguajes.
- Código con seguridad de tipos, lo que significa que únicamente se accede a los tipos de forma permisible y bien definida.
- Las herramientas que gestionan y permiten una ejecución del código de alto rendimiento.

### Tipo valor vs. Tipo referencia

El sistema de tipos comunes soporta dos categorías generales de tipos: *tipos valor* y *tipos referencia*.

Una variable de tipo valor contiene directamente sus datos. Cada variable de tipo valor tiene su propia copia de datos, de modo que las operaciones en una variable de tipo valor no pueden afectar a otra variable.

Una variable de tipo referencia contiene una referencia o puntero al valor de un objeto. Dos variables de tipo referencia pueden referirse al mismo objeto, de modo que las operaciones en una variable de tipo referencia pueden afectar al objeto referenciado por otra variable de tipo referencia.

## Tipos de datos

Tipo Visual Basic .NET	Tamaño de almacenamiento	Rango de valores
Boolean	2 bytes	Verdadero o Falso
Date	8 bytes	0:00:00 del 1 de enero de 0001 a 11:59:59 PM del 31 de diciembre de 9999
Decimal	16 bytes	Hasta 29 dígitos significativos, con valores de hasta $9,228 \times 10$ (con signo)
Double	8 bytes	-4,94065645841246544E-324 a +1,79769313486231570E+308 (con signo)
Integer	4 bytes	-2.147.483.648 a +2.147.483.647 (con signo)
Single	4 bytes	-3,4028235E+38 a 1,401298E-45 (con signo)
String	Varía	0 a 2.000 millones aproximadamente de caracteres Unicode

\*\*\*\*\*

### Introducción

La ilustración anterior lista algunos de los tipos más utilizados en Visual Basic .NET. El término *con signo* significa que los números pueden ser positivos o negativos, como en +48 o -48.

---

**Nota** Encontrará una lista completa de los tipos de datos de Visual Basic .NET y sus descripciones en la documentación de Visual Basic .NET, realizando una búsqueda por “Resumen de Tipos de Datos”.

---

## Cómo escoger un tipo de datos

Escoger tipo de datos...	para gestionar...	Tipo CTS	Ejemplo
Boolean	Condiciones de Verdadero o Falso	Valor	Verdadero
Short, Integer, Long, Byte	Enteros	Valor	23 (Entero)
Single, Double, Decimal	Números con enteros y partes de fracciones	Valor	9456,72 (Decimal)
Date	Valores fecha y hora	Valor	02/12/2003 12:30:42 A.M.
String	Caracteres imprimibles y visualizables en pantalla	Referencia	"Casa"
Object	Un puntero al valor de un objeto	Referencia	myClass myPerson

\*\*\*\*\*

### Introducción

Visual Basic no requiere que seleccione explícitamente un tipo de datos cuando declara una variable. Sin embargo, es una buena idea hacerlo, ya que de esta forma sus aplicaciones serán más fiables y requerirán menos memoria. El tipo de datos determina los valores permitidos para una variable, que, a su vez, determinan las operaciones que pueden realizarse sobre esa variable.

### Seleccionar un tipo de datos

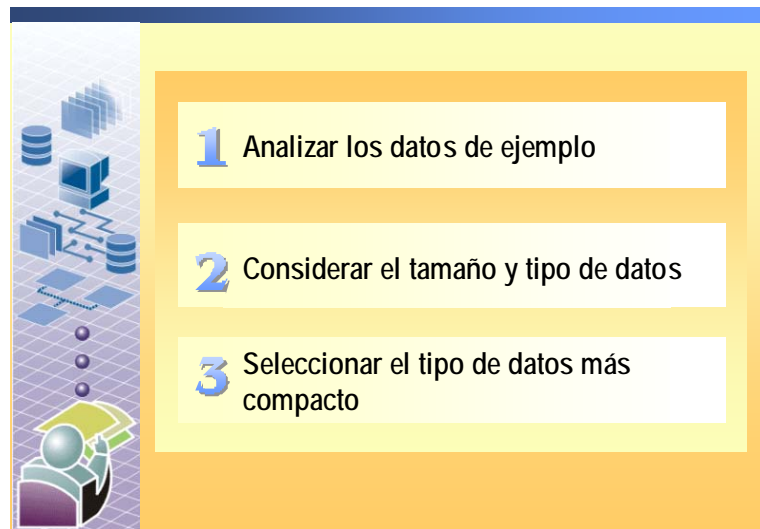
Puede seguir las directrices generales que se muestran en la ilustración anterior para seleccionar el tipo de datos adecuado para una variable.

### Tipado fuerte

Si usted declara variables sin tipo de datos, se les otorga de forma predeterminada el tipo de datos **Object**. Esto facilita la escritura rápida de programas, pero puede hacer que éstos se ejecuten más lentamente. La especificación de tipos de datos para todas sus variables se denomina *tipado fuerte*. Se recomienda el uso de tipado fuerte porque:

- Minimiza la cantidad de memoria que utiliza la aplicación.
- Permite al compilador realizar la verificación de tipos. Este proceso detecta las instrucciones que pueden fallar en tiempo de ejecución debido a variables y valores que no concuerdan.
- La ejecución del código es más rápida.
- Permite el soporte de Microsoft IntelliSense® en sus variables. Esta tecnología permite ver sus propiedades y demás miembros mientras escribe el código.

## Práctica: escoger tipos de datos



\*\*\*\*\*

**Introducción** Trabajar en parejas para analizar los datos de ejemplo y elegir el tipo de datos adecuado.

**Instrucciones** Seleccionar el tipo de datos más compacto para cada uno de los siguientes ejemplos de datos:

- Dirección

**String.**

---



---

- Cantidad de un préstamo

**Decimal. Utilizar el tipo Decimal para moneda porque los decimales no se completan. Singles y Dobles pueden completarse.**

---



---

- Número de teléfono

**String.**

---



---

- Tipo de interés

**Single.**

---



---



- Cumpleaños

**Date.**

---

---

- Número de identificación personal

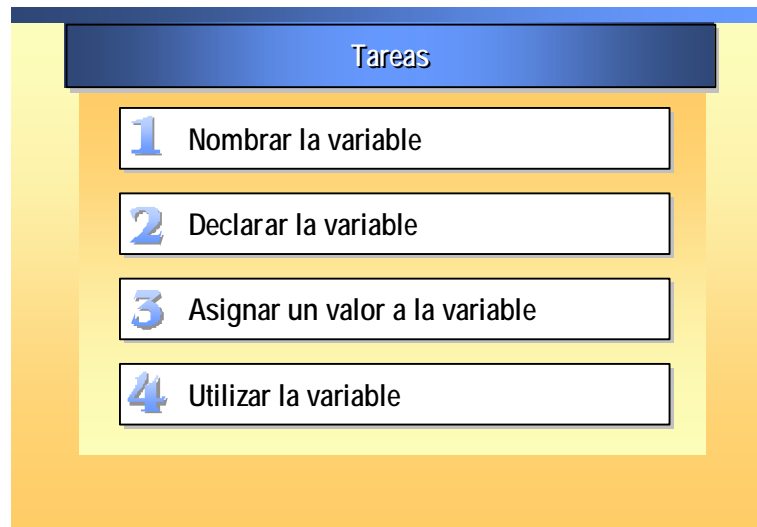
**String:** cuando las reglas de negocio requieran una mezcla de caracteres alfabéticos y numéricos.

**Entero, Short o Long** (el tipo de datos numérico apropiado más pequeño): cuando no sean necesarios caracteres alfabéticos.

---

---

## Lección: Uso de variables



\*\*\*\*\*

**Introducción** Antes de que pueda utilizar una variable en su aplicación, necesita nombrar, declarar la variable y asignarle un valor. Esta lección explica cómo crear y utilizar variables y constantes.

**Estructura de la lección** Esta lección incluye los siguientes temas y actividades:

- ¿Qué son las variables?
- Cómo nombrar las variables
- Cómo declarar las variables
- Cómo afecta **Option Explicit** a las variables
- Cómo asignar valores a las variables
- Cómo utilizar las variables
- Variables vs. Constantes
- Práctica: Encontrar los errores

**Objetivos de la lección** En esta lección, aprenderá a:

- Explicar el objetivo de las variables.
- Seguir las reglas y directrices establecidas para nombrar variables.
- Declarar variables de tipo valor y de tipo referencia.
- Asignar valores a variables.
- Utilizar variables en una aplicación.
- Describir la diferencia entre variables y constantes.

## ¿Qué son las variables?

- Las variables almacenan valores que pueden cambiar cuando una aplicación se está ejecutando
- Las variables tienen seis elementos básicos:

Elemento	Descripción
Nombre	La palabra que identifica a la variable en código
Dirección	La ubicación de memoria donde se almacena el valor
Tipo de datos	El tipo y tamaño inicial de datos que la variable puede almacenar
Valor	El valor en la dirección de la variable
Ámbito	El conjunto de todo el código que puede acceder y utilizar la variable
Vida	El intervalo de tiempo durante el cual una variable es válida

\*\*\*\*\*

### Introducción

A menudo, cuando realizamos cálculos en Visual Basic necesitamos almacenar valores temporalmente. Por ejemplo, es posible que necesitemos calcular varios valores, compararlos y realizar diferentes operaciones con ellos dependiendo del resultado de la comparación.

### Definición

Utilizamos variables para almacenar valores que pueden cambiar cuando una aplicación se está ejecutando.

### Elementos de las variables

Una variable tiene los seis elementos siguientes:

Elemento	Descripción
Nombre	La palabra que utilizamos para hacer referencia a la variable en código.
Dirección	La ubicación de memoria donde se almacena el valor de la variable.
Tipo de datos	El tipo y tamaño inicial de datos que la variable puede almacenar.
Valor	El valor en la dirección de la variable.
Ámbito	El conjunto de todo el código que puede acceder y utilizar la variable.
Tiempo de vida	El intervalo de tiempo durante el cual una variable es válida y está disponible para poder ser utilizada. Su valor puede cambiar durante su vida, pero siempre mantiene alguno mientras existe en memoria.

### Ejemplos de variables

Una variable puede utilizarse de muchas formas, incluyendo las siguientes:

- Como contador que almacena el número de veces en que un evento se produce
- Como almacenamiento temporal para valores de propiedades
- Como ubicación para guardar un valor de retorno de una función
- Como ubicación para almacenar nombres de directorio o archivos

## Cómo nombrar las variables

- Reglas para poner nombres
  - Empezar con un carácter alfabético o guión bajo
  - No utilizar espacios ni símbolos
  - No utilizar palabras clave como **Integer**
- Ejemplos de nombres de variables
  - **NombreCl iente** (PascalCasing)
  - **numeroCuenta** (camelCasing)

\*\*\*\*\*

### Introducción

Cuando declaramos una variable, es importante desarrollar una estrategia de asignación de nombres. Tanto la claridad como la coherencia son importantes, especialmente cuando otros necesitarán leer o mantener nuestro código.

### Reglas

Cuando nombramos una variable en Visual Basic .NET, debemos tener en cuenta las siguientes reglas:

- Iniciar cada nombre de variable con un carácter alfabético o un guión bajo (\_).
- No utilizar espacios ni símbolos.
- No utilizar palabras clave como **Integer** o **Date**.

### Recomendaciones

Se recomienda que tenga en cuenta las siguientes directrices cuando nombre variables:

- Proporcione nombres descriptivos y con significado, como *numeroCuenta*. Aunque escribir un nombre largo de variable puede resultar tedioso en el momento de escribir código, hará que el código sea más fácil de leer y mantener.
- Inicie cada palabra del nombre de una variable pública con letra mayúscula, como *NombreCliente*. Esto se denomina *PascalCasing*.
- Evite el uso de abreviaturas. Aunque es posible utilizar un nombre de variable como *nocta*, el código será mucho más fácil de leer si utilizamos el nombre *numeroCuenta*. En caso de ser necesario el uso de abreviaturas, debemos asegurarnos de que sean coherentes en toda la aplicación.
- Utilice un nombre único dentro del ámbito de la variable. *Ámbito (scope)* hace referencia al subconjunto del código que reconoce la variable.
- Cuando declare variables locales y privadas, inicie la primera palabra con un carácter en minúscula, como en *nuevoCliente*. Esto se denomina *camelCasing*.

## Cómo declarar variables

- Sintaxis para declarar variables
  - `Dim nombreVariable As Type`
- Ejemplos de variables de tipo valor

```
Dim numberBooks As Integer
Dim squareFootage As Single
```

- Ejemplos de variables de tipo referencia

```
Dim myForm As Form
Dim userInput As String
```

\*\*\*\*\*

### Introducción

Declaramos una variable para especificar su nombre y características. La instrucción de declaración tanto para variables de tipo valor como de tipo referencia es la instrucción **Dim**. La ubicación y el contenido de la declaración determinan las características de la variable.

### Sintaxis

Para declarar una variable, utilice la siguiente sintaxis:

```
Dim nombreVariable As Type
```

Utilizamos la instrucción **Dim** para declarar y asignar espacio de almacenamiento para variables en bloques, procedimientos, módulos, estructuras y clases. Utilizamos la cláusula **As** en la instrucción **Dim** para especificar el tipo de datos de la variable.

---

**Nota** La palabra clave **Dim** es una abreviatura de la palabra *dimension*.

---

### Ejemplos de tipos valor

Los siguientes ejemplos muestran cómo declarar las variables con tipos valor predefinidos:

```
Dim numberBooks As Integer
Dim squareFootage As Single
```

### Ejemplos de tipos referencia

Los siguientes ejemplos muestran cómo declarar las variables con tipos referencia:

```
Dim myForm As Form
Dim userInput As String
```

Aunque la sintaxis para declarar variables de tipo valor y de tipo referencia es similar, el entorno de ejecución los gestiona de modo distinto. Una variable de tipo referencia siempre contiene un puntero a un valor de ese tipo o una referencia nula. Una variable de tipo valor contiene el valor real de la variable.

## Cómo afecta Option Explicit a las variables

- **Option Explicit habilitado (predeterminado)**
  - Obliga a declarar explícitamente las variables antes de utilizarlas
  - Reduce errores lógicos y facilita el mantenimiento del código
  - Produce una ejecución del código más rápida
- **Option Explicit no habilitado**
  - Permite utilizar implícitamente variables sin declararlas
  - Aumenta la probabilidad de conflictos de nombres y comportamiento imprevisto debido a errores de ortografía
  - Produce una ejecución del código más lenta

\*\*\*\*\*

### Introducción

En general, debería declarar explícitamente las variables de su aplicación antes de utilizarlas. De este modo, se reduce la probabilidad de errores lógicos y se facilita el mantenimiento del código. Aunque no es recomendable, puede utilizar variables en su aplicación sin antes declararlas. Este proceso se denomina declaración *implícita*.

### Cómo funciona Option Explicit

Cuando **Option Explicit** está **On** (de forma predeterminada está habilitado), debe declarar explícitamente variables antes de poder utilizarlas, de lo contrario el compilador generará un error.

Cuando **Option Explicit** no está habilitado (**Off**), puede declarar implícitamente una variable simplemente utilizándola en su código. Se creará como un tipo objeto. Aunque puede resultar conveniente declarar variables implícitamente, se incrementa la probabilidad de conflictos de nombres y de un comportamiento imprevisto debido a errores de ortografía. Además, genera un uso ineficaz del almacenamiento de memoria.

### Ejemplo de una ventaja de Option Explicit

Por ejemplo, supongamos que la variable *sueldoActual* guarda el salario anual de un empleado. Un procedimiento utiliza esta variable en una fórmula que calcula el bonus del empleado. Supongamos que la variable *sueldoActual* se escribe incorrectamente, como se muestra en el siguiente ejemplo:

```
Dim sueldoActual As Integer
bonusActual = sueldoActual * .10
```

Con **Option Explicit** no habilitado (**Off**), este cálculo supondrá un bonus de \$0.00 debido a que la variable *sueldoActual* se declarará implícitamente como una nueva variable Object y se inicializará vacía. Cuando *sueldoActual* se utilice en el cálculo, Visual Basic lo convertirá automáticamente a 0. Si este cálculo se realiza varias veces en una estructura cíclica, la aplicación se ejecutará sustancialmente más lentamente porque Visual Basic necesita tiempo para crear, inicializar, y convertir la variable cada vez.

## Configurar Option Explicit

Puede establecer **Option Explicit** como **On** u **Off** a nivel de proyecto del entorno de desarrollo.

### ⚡ Configurar Option Explicit

1. En el Explorador de soluciones, haga clic en el nombre del proyecto para el que desea configurar **Option Explicit**.
2. En el menú **Ver**, haga clic en **Páginas de propiedades**.
3. Expanda la carpeta **Propiedades comunes** y, a continuación, haga clic en la carpeta **Generar**.
4. Bajo **Valores predeterminados del compilador**, haga clic en **Off** u **On** según lo deseado en la lista **Option Explicit** y, a continuación, haga clic en **OK**.

También puede configurar **Option Explicit** como **On** u **Off** mediante la instrucción adecuada al inicio de su código:

```
' This turns Option Explicit On
Option Explicit On
' This turns Option Explicit Off
Option Explicit Off
```

## Cómo asignar valores a las variables

- Podemos:
- Asignar un valor a una variable después de declararla
- Asignar un valor a una variable mientras la declaramos

```
Dim cumpleaños As Date  
cumpleaños = #3/9/1974#
```

```
Dim cumpleaños As Date = #3/9/1974#
```

\*\*\*\*\*

Introducción	Antes de poder utilizar variables en su aplicación, debe asignarles un valor. Puede asignar un valor a una variable después de declararla o mientras la declara.
Sintaxis	<p>Para asignar un valor a una variable, utilice el operador de asignación (=), como se muestra en la siguiente expresión:</p> <p><i>NombreVariable = Valor</i></p> <p>El valor del lado derecho de la expresión se asigna a la variable del lado izquierdo de la expresión.</p>
Asignar un valor después de declarar	<p>Puede asignar valores a variables después de declararlas, como se muestra en el siguiente ejemplo:</p> <pre>Dim cumpleaños As Date cumpleaños = #3/9/1974#</pre>
Asignar valores mientras se declara	<p>Cuando creamos una variable con la instrucción <b>Dim</b>, Visual Basic inicializa automáticamente las variables numéricas a 0, las cadenas de texto a vacías ("") y las variables de fecha a 1 de enero de 0001.</p> <p>También puede asignar un valor a una variable mientras la declara, como se muestra en los siguientes ejemplos:</p> <pre>Dim cumpleaños As Date = #3/9/1974# Dim goodNews As String = "Su cheque está en el correo." Dim testCondition As Boolean = True</pre>

---

**Nota** Los valores de fecha (**Date**) deben estar encerrados entre almohadillas (##), y los valores de cadena (**String**) deben estar encerrados entre comillas (").

---



## Cómo utilizar variables

Podemos utilizar variables para:

- Almacenar valores de expresiones
- Almacenar entrada del usuario
- Almacenar objetos
- Almacenar valores de propiedades
- Devolver valores
- Mostrar la salida

\*\*\*\*\*

### Introducción

Después de nombrar y declarar variables y asignarles un valor, puede empezar a utilizarlas en su aplicación. Las variables pueden mantener el mismo valor en toda una aplicación o pueden cambiar de valor varias veces, dependiendo de cómo las utilicemos.

### Ejemplos

Podemos utilizar variables para almacenar datos originados por una expresión, como se muestra en los siguientes ejemplos:

```
newMessage = "¡Tiene correo nuevo!"  
unreadMail = totalMail - readMail
```

Podemos utilizar variables para almacenar la entrada de información por el usuario, como se muestra en los siguientes ejemplos:

```
userName = nameTextBox.Text  
applicationDate = appDateTextBox.Text
```

Podemos utilizar variables para almacenar objetos, como se muestra en el siguiente ejemplo:

```
myForm = mainForm
```

## Variables vs. Constantes

Variables	Constantes
Declarar con Dim	Declarar con Const
Los valores <b>cambian</b> mientras se ejecuta la aplicación	Los valores <b>no cambian</b> mientras se ejecuta la aplicación
Utilizan <b>más</b> memoria que las constantes	Utilizan <b>menos</b> memoria que las variables

Sintaxis para declarar una constante:  
**Const *constantName* As Type**

\*\*\*\*\*

### Introducción

Si una variable de su programa contiene un valor que no cambia nunca, considere almacenar el valor como una constante en lugar de una variable. Las constantes proporcionan una forma de utilizar nombres con significado en lugar de un valor que no cambia (como  $\pi$ , una cantidad matemática fija).

### Cómo funcionan las constantes

Las constantes almacenan valores que, como su nombre indica, permanecen constantes durante la ejecución de una aplicación. Algunas ventajas de utilizar constantes son:

- Hacen que el código sea más fácil de leer.
- Utilizan menos memoria que las variables.
- Hacen que los cambios a nivel de aplicación sean más fáciles de implementar.

### Sintaxis

Para declarar una constante, utilice la instrucción **Const** con la siguiente sintaxis:

```
Const nombreConstante As Type
```

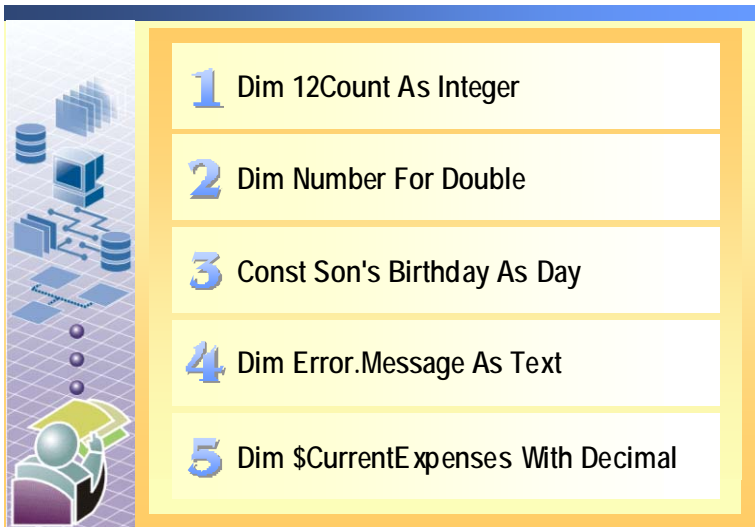
Nombramos y especificamos los niveles de ámbito de las constantes siguiendo las mismas reglas que para las variables.

### Ejemplo

El siguiente ejemplo muestra cómo declarar y utilizar una constante:

```
Dim area, radio, circunferencia As Double
Const Pi As Double = 3.1415
area = Pi * radio ^ 2
circunferencia = 2 * Pi * radio
```

## Práctica: Encontrar errores



\*\*\*\*\*

### Introducción

Trabajar en parejas para encontrar y arreglar los errores de la diapositiva.

### Instrucciones

Volver a escribir las siguientes instrucciones declarativas para corregir los errores. Seguir las reglas y recomendaciones de nomenclatura para variables locales.

```
Dim 12Count As Integer
Dim Number For Double
Const Son's Birthday As Day
Dim Error.Message As Text
Dim $CurrentExpenses With Decimal
```

**Dim count As Integer**

**Dim number As Double**

**Const sonBirthday As Date**

**Dim errorMessage As String**

**Dim currentExpenses As Decimal**

---

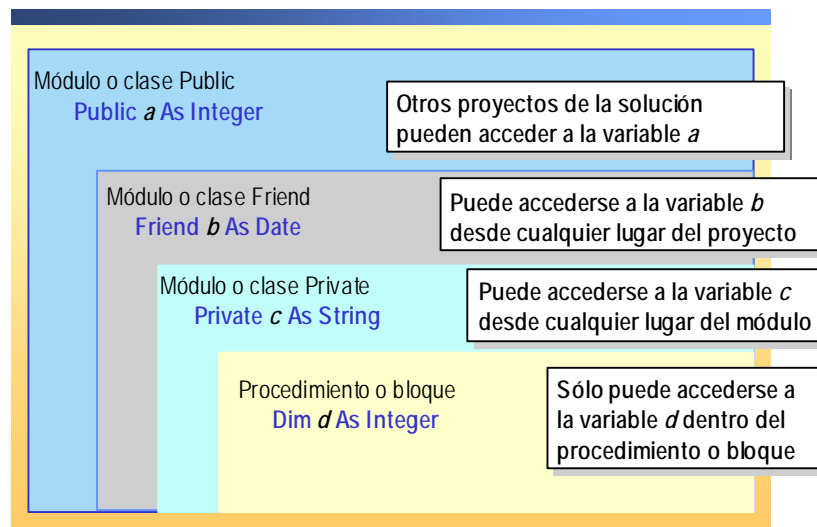
---

---

---

---

## Lección: ámbito de una variable



\*\*\*\*\*

### Introducción

Cuando utilizamos variables, debemos asegurarnos de que son accesibles desde todas las áreas de código que hacen referencia a ellas. Por otra parte, es posible que necesite restringir el acceso a determinadas variables. Todo el conjunto de código que puede hacer referencia a una variable por su nombre se denomina *ámbito (scope)* de la variable. Esta lección describe los diferentes niveles de ámbito aplicables a variables y explica cómo asegurarse de que cada variable de nuestra aplicación tiene el ámbito adecuado.

### Estructura de la lección

Esta lección incluye los siguientes temas y actividades:

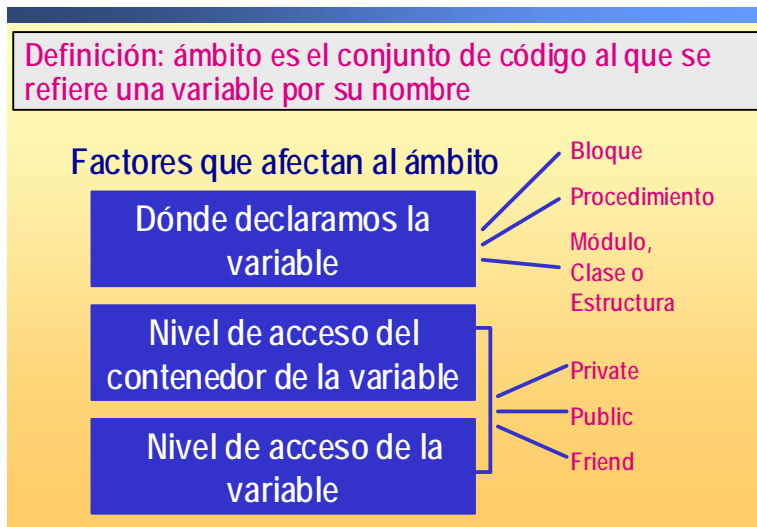
- ¿Qué es el ámbito?
- Cómo declarar variables locales
- Cómo declarar variables estáticas
- Cómo declarar variables de módulo
- Demostración multimedia: cómo configurar los niveles de acceso para las variables
- Práctica: configurar niveles de acceso para variables

### Objetivos de la lección

En esta lección, aprenderá a:

- Explicar los diferentes niveles de ámbito para las variables.
- Escoger el nivel de acceso adecuado para una variable, basándose en su uso dentro de una aplicación.
- Declarar variables locales en bloques y procedimientos.
- Declarar variables estáticas locales.
- Declarar variables de módulo para su uso en módulos estándares, clases, proyectos, soluciones y espacios de nombres.

## ¿Qué es el ámbito?



\*\*\*\*\*

### Introducción

Cuando declaramos variables, uno de los primeros aspectos que probablemente consideraremos es el ámbito. Si utilizamos una variable fuera de su ámbito, el compilador generará un error.

### Definiciones

Para estudiar el modo de trabajo del ámbito con variables, debemos estar familiarizados con los términos y definiciones de la siguiente tabla:

Término	Definición
Ámbito de la variable	Conjunto de código al que se refiere una variable por su nombre asignado sin cualificadores.
Bloque	Bloque de código que empieza con una condición como <b>If</b> o <b>While</b> y termina con una instrucción <b>End</b> , <b>Loop</b> o <b>Next</b> .
Procedimiento	Bloque de código que empieza con una instrucción de declaración como <b>Sub</b> y termina con una instrucción <b>End</b> .
Módulo	Archivo que agrupa procedimientos comunes y datos globales para hacerlos accesibles a su uso en uno o más proyectos.
Ensamblado	Un archivo ejecutable independiente en el que se compilan los diversos archivos cuando genera una solución.
Modificador de acceso	Palabra clave como <b>Public</b> o <b>Friend</b> que utilizamos para especificar el nivel de acceso de una variable o su contenedor (módulo, clase o estructura).

**Factores que afectan al ámbito**

Asignamos el ámbito de una variable cuando la declaramos. Existen tres factores principales que afectan al ámbito de una variable:

- Dónde la declaramos: dentro de un bloque, procedimiento, módulo, clase o estructura.
- El nivel de acceso (**Public**, **Friend** o **Private**) del módulo, clase o estructura en que se declara. El ámbito de una variable no puede exceder el ámbito de su contenedor.
- La sintaxis que utilizamos para declarar la variable (**Dim**, **Private**, **Friend** o **Public**).

**Niveles de ámbito**

Una variable puede tener uno de los siguientes niveles de ámbito:

Nivel de ámbito	Descripción
Bloque	Disponible únicamente dentro del bloque de código en el que se declara
Procedimiento	Disponible únicamente dentro del procedimiento en el que se declara
Módulo	Disponible para todo el código del módulo, clase o estructura en el que se declara
Espacio de nombres	Disponible para todo el código del espacio de nombres

Podemos definir todavía más el ámbito de una variable de módulo declarando un modificador de acceso (**Private**, **Public** o **Friend**) cuando la declaramos. Estudiaremos más sobre los modificadores de acceso en la sección *Cómo declarar variables de módulo* de esta lección.

---

**Nota** Si desea aprender más sobre ámbitos, consulte la documentación de Visual Basic .NET.

---

## Cómo declarar variables locales

Dónde declarar	Palabra clave	Modificador de acceso	Ámbito
En bloque	Dim	Ninguno	Nivel bloque
En procedimiento	Dim	Ninguno	Nivel procedimiento

### Ejemplo de variable local: a nivel de bloque

```
If x <> 0 Then
    Dim blockNumber As Integer
    blockNumber = x + 1
End If
```

### Ejemplo de variable local: a nivel de procedimiento

```
Sub ShowMessage_Click( )
    Dim miVariable As String
    ' Insert code to add functionality
End Sub
```

\*\*\*\*\*

#### Introducción

Cuando declaramos variables en un bloque o procedimiento, nos referimos a *variables locales* y significa que su ámbito está limitado al bloque o procedimiento en el que se declaran. Si consideramos el ámbito, las variables locales son una buena elección para cualquier tipo de cálculo temporal. Utilizan memoria sólo cuando su procedimiento se está ejecutando, y sus nombres no son susceptibles de conflictos de nomenclatura.

#### Sintaxis

Para declarar una variable local, utilice la instrucción **Dim** con la siguiente sintaxis:

```
Dim nombreVariable As Type
```

#### Ejemplo de ámbito a nivel de bloque

El siguiente ejemplo muestra cómo declarar una variable local denominada *blockNumber* con ámbito a nivel de bloque:

```
If x <> 0 Then
    Dim blockNumber As Integer
    blockNumber = 1 / x
End If
```

---

**Nota** En general, cuando declaramos cualquier variable, es una buena práctica de programación mantener el ámbito lo más estrecho posible (el ámbito de bloque es el más estrecho). Mantener el ámbito estrecho ayuda a conservar memoria y minimiza las posibilidades de que el código haga referencia a una variable equivocada.

---

**Ejemplo de ámbito a nivel de procedimiento**

El siguiente ejemplo muestra cómo declarar una variable local *name* con ámbito a nivel de procedimiento:

```
Sub ShowMessage_Click( )  
    Dim name As String  
    name = NameTextBox.Text  
    MessageBox.Show("Bienvenido de nuevo, " & name & "!")  
End Sub
```

En este ejemplo, el contenido del cuadro de texto **NameTextBox** se asigna a la variable *name*, y la variable se utiliza como parte del texto en el cuadro de mensaje.



## Cómo declarar variables estáticas

- Dónde: declarar dentro de un bloque o procedimiento
- Sintaxis: utilizar la palabra clave Static (no modificador de acceso)
  - *Static nombreVariable As Type*
- Ejemplo

```
Sub AddItem_Click( )
    Static items As Integer
    'Añadir 1 al contador
    items += 1
    MessageBox.Show ("El contador es ahora " & items)
End Sub
```

\*\*\*\*\*

### Introducción

La vida de una variable local empieza cuando un procedimiento la invoca y acaba cuando el procedimiento finaliza. Cuando acaba la vida de una variable local, la instancia de la variable se destruye y su valor se pierde. En ocasiones, es posible que deseemos que la vida de una variable local sea más larga que la vida del procedimiento. Por ejemplo, podemos desear que un procedimiento realice una acción específica la primera vez que sea invocado y que no haga nada en las siguientes llamadas. Podemos declarar una variable estática para conseguir esta funcionalidad.

### Definición

Una *variable estática* perdurará mientras la aplicación siga ejecutándose. Las variables estáticas siguen existiendo y conservan sus últimos valores entre invocaciones al procedimiento en el que se han declarado. Sin embargo, el código de otros procedimientos no puede acceder a ellas.

### Sintaxis

Para declarar una variable estática local, utilizaremos la siguiente sintaxis:

```
Static variableName As Type
```

**Ejemplo**

Las variables estáticas resultan útiles para mantener contadores que se utilizan únicamente dentro de un procedimiento. El siguiente ejemplo muestra cómo declarar una variable estática que realiza el seguimiento de cuántas veces se ejecuta el procedimiento **AddItem**. Cada vez que el procedimiento se ejecuta, el valor almacenado en *items* se incrementa en 1.

```
Sub AddItem_Click( )  
    Static items As Integer  
  
    ' Añadir 1 al contador  
    items += 1  
  
    ' Usar ampersand(&) para combinar una cadena y una  
variable  
    MessageBox.Show("El contador es ahora " & items)  
End Sub
```

---

**Nota** Puede conseguir los mismos resultados que con el código anterior declarando *items* como una variable de módulo. Sin embargo, otros procedimientos tendrían acceso a la variable y podrían cambiarla. A menos que se desee ámbito a nivel de módulo, es recomendable utilizar una variable estática.

---

## Cómo declarar variables de módulo

### ■ Declarar en un módulo, clase o estructura

Utilizar modificador de acceso	Ámbito
Private	Módulo
Friend	Proyecto
Public	Solución

### ■ Ejemplos

```
Private myModuleMessage As String
Friend myProjectMessage As String
Public mySolutionMessage As String
```

\*\*\*\*\*

#### Introducción

Las variables declaradas en un módulo, clase o estructura pero no dentro de un procedimiento se denominan *variables de módulo*. Después de declarar una variable de módulo, podemos asignarle un valor, asignar su ámbito utilizando un modificador de acceso y utilizarla dentro de ese ámbito.

#### Sintaxis

Para declarar una variable de módulo, utilice la siguiente sintaxis:

```
AccessModifier nombreVariable As Type
```

Existen varios modificadores de acceso, incluyendo los descritos en la siguiente tabla:

Modificador de acceso	Ámbito	Descripción
<b>Private</b>	Módulo	Accesible desde cualquier lugar del módulo, clase o estructura en el que se declara. Si declaramos una variable de módulo con la palabra clave <b>Dim</b> , el acceso predeterminado es <b>Private</b> .
<b>Friend</b>	Proyecto	Accesible desde cualquier lugar del proyecto pero no fuera del mismo.
<b>Public</b>	Solución	Accesible desde cualquier lugar de la solución. No hay ninguna restricción en el uso de variables <b>Públicas</b> .

**Nota** El ámbito de una variable de módulo está determinado no sólo por el modificador de acceso utilizado para declararla, sino también por el nivel de acceso del módulo, clase o estructura en el que se declara. El ámbito de la variable no puede ser mayor que el ámbito de su contenedor. En la documentación de Visual Basic .NET, encontrará más información sobre accesibilidad y una lista completa de los modificadores de acceso.

**Ejemplo de ámbito a nivel de módulo**

El siguiente ejemplo muestra cómo declarar y utilizar una variable con el ámbito a nivel de módulo:

```
' Put the following declaration at module level
' (not in any procedure)
Private myModuleMessage As String
' ...
Sub InitializeModuleVariable( )
    myModuleMessage = "This variable has module-level scope."
End Sub
' ...
Sub UseModuleVariable( )
    MessageBox.Show(myModuleMessage)
End Sub
```

**Ejemplo de ámbito a nivel de proyecto**

El siguiente ejemplo muestra cómo declarar y utilizar una variable con el ámbito a nivel de proyecto:

```
' Put the following declaration at module level
' (not in any procedure)
Friend MyProjectMessage As String
' ...
Sub InitializeProjectVariable( )
    MyProjectMessage = "This variable has project-level scope."
End Sub
' ...
Sub UseProjectVariable( )
    MessageBox.Show(MyProjectMessage)
End Sub
```

**Ejemplo de ámbito a nivel de solución**

El siguiente ejemplo muestra cómo declarar y utilizar una variable con el ámbito a nivel de solución:

```
' Put the following declaration at module level
' (not in any procedure)
Public MySolutionMessage As String
' ...
Sub InitializeSolutionVariable( )
    MySolutionMessage = _
        "This variable has solution-level scope."
End Sub
' ...
Sub UseSolutionVariable( )
    MessageBox.Show(MySolutionMessage)
End Sub
```

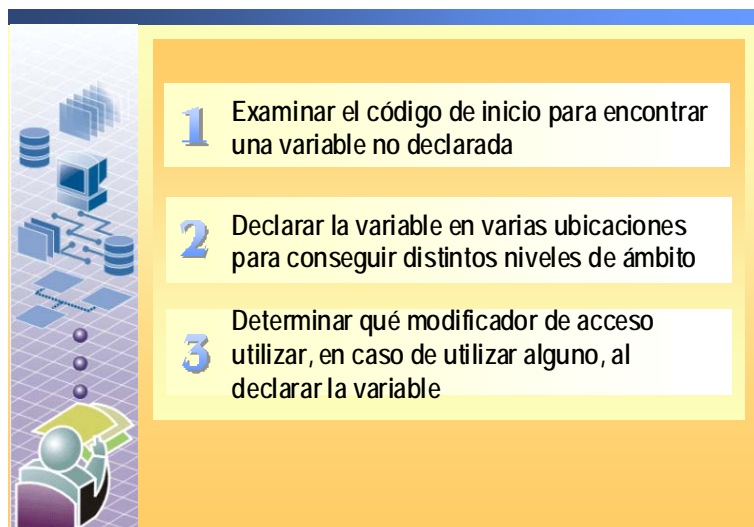
## Multimedia: Cómo configurar los niveles de acceso para las variables

\*\*\*\*\*

### Introducción

Esta demostración multimedia muestra cómo declarar una variable con diferentes niveles de ámbito. Veremos dónde declarar variables y cómo especificar los modificadores de acceso adecuados para asignar varios niveles de ámbito distintos.

## Práctica: Configurar niveles de acceso para variables



\*\*\*\*\*

### Introducción

En esta práctica, examinaremos el código de ejemplo para encontrar una variable que no ha sido declarada. Determinaremos dónde declarar la variable y qué modificador de acceso utilizar para asignar distintos niveles de ámbito a la variable.

### Instrucciones

#### 🔍 Abrir el código de inicio para la práctica

1. Abra Visual Studio .NET.
2. Abra el archivo Scope.sln file, que se encuentra en la carpeta Scope\Starter dentro del fichero practs03.zip.

#### 🔍 Establecer niveles de acceso para la variable

1. Abra el Editor de código de Form1.vb.
2. Encuentre la variable no declarada *myVariable* en el código de inicio.
3. Declare *myVariable* como una variable local con ámbito de nivel de procedimiento.

---

**Nota** El entorno de desarrollo resalta los errores de código mediante un subrayado en azul. Si ve un error, pose el cursor del ratón sobre el código que contiene el error y aparecerá una sugerencia que le ayudará a corregirlo.

---

4. Declare *myVariable* como una variable de módulo disponible en cualquier lugar de la clase **Form1**.
5. Declare *myVariable* como una variable de módulo disponible en cualquier lugar del proyecto.
6. Declare *myVariable* como una variable de módulo disponible en cualquier lugar de la solución.

### Archivos de solución

Los archivos de solución de esta práctica se encuentran en la carpeta Scope\Solution dentro del fichero practs03.zip.

## Lección: Convertir tipos de datos

- ¿Cuáles son las funciones de conversión?
- Cómo convertir explícitamente tipos de datos
- Cómo funciona la conversión de datos implícita

\*\*\*\*\*

### Introducción

El proceso de convertir el valor de un tipo de datos en otro se denomina *conversión o casting*. Podemos convertir explícitamente valores de un tipo en otro antes de que sean utilizados, evitando así errores en nuestro código y haciendo que éste se ejecute más rápidamente. Visual Basic .NET también puede realizar algunas conversiones de tipos de datos automática o implícitamente, como convertir algunas cadenas a enteros. Sin embargo, las conversiones implícitas pueden producir resultados imprevistos.

En esta lección, estudiaremos las diferencias entre la conversión explícita y la implícita. También estudiaremos cómo utilizar las funciones de conversión de Visual Basic .NET para convertir explícitamente tipos de datos.

### Estructura de la lección

Esta lección incluye los temas siguientes:

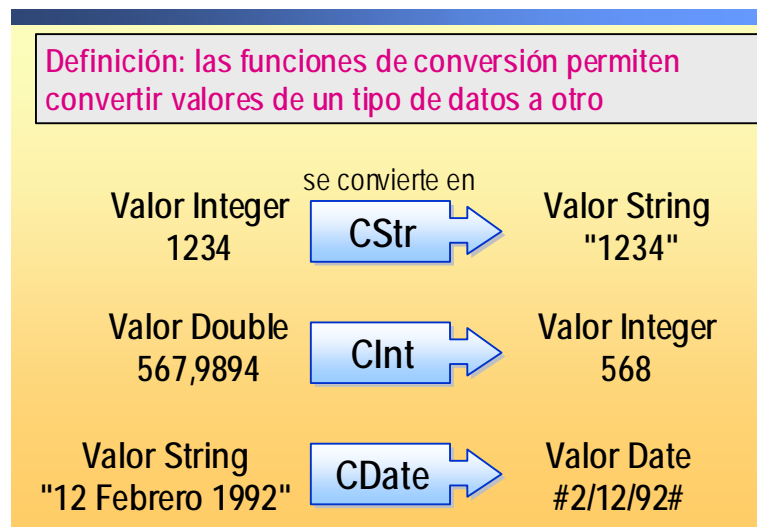
- ¿Cuáles son las funciones de conversión?
- Cómo convertir explícitamente tipos de datos
- Cómo funciona la conversión implícita de datos

### Objetivos de la lección

En esta lección, aprenderá a:

- Describir algunas funciones de conversión habituales.
- Convertir variables explícitamente de un tipo de datos en otro.
- Explicar el funcionamiento de la conversión de datos implícita y cuándo no utilizarla.

## ¿Cuáles son las funciones de conversión?



\*\*\*\*\*

Introducción	Podemos utilizar funciones de conversión para forzar el resultado de una operación a un tipo de datos particular en lugar del tipo de datos por defecto. Por ejemplo, podemos convertir un valor de tipo cadena a un valor entero.
Definición	Las <i>funciones de conversión</i> permiten convertir explícitamente un valor de un tipo de datos a otro.
Ejemplos de funciones de conversión	Visual Basic ofrece una amplia lista de funciones de conversión, incluyendo las descritas en la siguiente tabla:

Función de conversión	Convierte en tipo de dato	Tipos de datos para conversión permitidos
<b>CStr</b>	<b>String</b>	Cualquier tipo numérico, <b>Boolean</b> , <b>Char</b> , <b>Date</b> , <b>Object</b>
<b>CInt</b>	<b>Integer</b>	Cualquier tipo numérico, <b>Boolean</b> , <b>String</b> , <b>Object</b>
<b>Cdbl</b>	<b>Double</b>	Cualquier tipo numérico, <b>Boolean</b> , <b>String</b> , <b>Object</b>
<b>CDate</b>	<b>Date</b>	<b>String</b> , <b>Object</b>
<b>CType</b>	Tipo especificado	El mismo tipo que el permitido para la función de conversión correspondiente



El código siguiente muestra cómo utilizar varias funciones de conversión:

```
Private Sub Button1_Click(...)
    ' Use string conversion on Integer

    Dim myVariable As Integer
    myVariable = 1234
    MessageBox.Show(myVariable)
    MessageBox.Show(CStr(myVariable) & "1")

End Sub

Private Sub Button2_Click(...)
    ' Use integer conversion on Double

    Dim myVariable As Double
    myVariable = 567.9894
    MessageBox.Show(myVariable)
    MessageBox.Show(CInt(myVariable))

End Sub

Private Sub Button3_Click(...)
    ' Use date conversion on String

    Dim myVariable As String
    myVariable = "February 12, 1992"
    MessageBox.Show(myVariable)
    MessageBox.Show(CDate(myVariable))

End Sub
```

## Cómo convertir explícitamente tipos de datos

Sintaxis: *NombreVariable = CFunction(Expression)*

### Ejemplo

- 1 Declarar una variable como tipo de datos String  
`Dim myString As String`
- 2 Declarar otra variable como tipo de datos Integer  
`Dim myInteger As Integer`
- 3 Asignar un valor a la variable string  
`myString = "1234"`
- 4 Convertir el valor string en un valor integer  
`myInteger = CInt(myString)`

\*\*\*\*\*

### Introducción

Se recomienda el uso de funciones de conversión para convertir valores explícitamente antes de que sean utilizados. Las conversiones explícitas se ejecutan más rápidamente que las conversiones implícitas porque no hay llamada a un procedimiento para llevar a cabo la conversión. Una *conversión implícita* es una conversión automática de un valor de un tipo en otro.

### Sintaxis

Para utilizar una función de conversión, se sitúa en el lado derecho de una instrucción de asignación, utilizando la siguiente sintaxis:

*VariableName = CFunction(Expression)*

El parámetro *expression* puede ser cualquier expresión válida, como una variable, un resultado de una función o un valor constante.

### Conversiones anchas frente a conversiones estrechas

Una *conversión ancha* cambia un valor a un tipo de datos que puede contener cualquier valor posible de los datos originales, como convertir un **Integer** en un **Long**. Una *conversión estrecha* cambia un valor a un tipo de datos que es posible que no pueda guardar algunos de los valores posibles, como cambiar un **String** en un **Integer**, o un **Long** en un **Integer**.

**Ejemplo**

El siguiente ejemplo muestra cómo utilizar las funciones de conversión **CStr** y **Cdbl** para escribir código que convierte pies y pulgadas A metros.

```
Private Sub Calculate_Click(...)
    Dim feet As Double, inches As Double
    Dim millimeters As Double, meters As Double

    'First, extract feet and inches from the text boxes.
    'The Text property returns a string, but we need a double,
    'so Cdbl( ) is used to perform the conversion.
    feet = Cdbl(FeetTextBox.Text)
    inches = Cdbl(InchesTextBox.Text)

    'Next, convert feet and inches to millimeters.
    millimeters = (inches * MillimetersPerInch) + _
        (feet * MillimetersPerInch * InchesPerFoot)

    'Convert millimeters to meters.
    meters = millimeters / 1000

    'Display the result in a label. Because the Text property
    'is a string, use CStr( ) to convert the double.
    MeterResult.Text = CStr(meters)
End Sub
```

En este ejemplo, podemos asumir que los valores constantes `MillimetersPerInch` e `InchesPerFoot` han sido declarados y se les han asignado valores adecuados a nivel de módulo.

**Ejemplo de CInt**

El siguiente ejemplo muestra cómo utilizar la función **CInt** para convertir un valor **Double** en un **Integer**:

```
Dim myDouble As Double
Dim myInt As Integer
myDouble = 2345.5678
' Set myInt to 2346
myInt = CInt(myDouble)
```

**Ejemplo de CDate**

El siguiente ejemplo muestra cómo utilizar la función **CDate** para convertir cadenas en valores **Date**. En general, no se recomienda codificar explícitamente fechas y horas como cadenas (como muestra este ejemplo). En lugar de ello, utilicemos literales de fecha y hora, como `#Feb 12, 1969#` y `#4:45:23 PM#`.

```
Dim myDateString, myTimeString As String
Dim myDate, myTime As Date
myDateString = "February 12, 1969"
myTimeString = "4:35:47 PM"
' ...
' Convert to Date data type
myDate = CDate(myDateString)
myTime = CDate(myTimeString)
```

## Cómo funciona la conversión de datos implícita

- Los tipos de datos son convertidos automáticamente
- No se requiere sintaxis especial en el código
- Ejemplo de conversión de datos implícita:

```
Dim sequence As String
Dim number As Integer
' ...
sequence = "1234"
number = sequence
' The value in sequence is implicitly converted
  to an Integer
```

- Desventajas de la conversión de datos implícita:
  - Puede producir resultados imprevistos
  - El código se ejecuta más lentamente
- Option Strict rechaza las conversiones implícitas de tipo estrechas

\*\*\*\*\*

### Introducción

Visual Basic puede realizar algunas conversiones de tipos de datos implícitamente. Una *conversión implícita* se produce cuando un valor se convierte automáticamente de un tipo de datos en otro requerido por el código en que el tipo de datos se utiliza. No requiere ninguna sintaxis especial en el código fuente.

### Ejemplos

Si colocamos la cadena “1234” en una variable **Integer**, Visual Basic convertirá automáticamente la cadena en un entero. O, si una cadena como “100” se añade a un valor numérico en la fórmula “100” + 10, Visual Basic convierte implícitamente la cadena en el valor entero 100 y lo añade a 10.

### Inconvenientes

Existen inconvenientes al depender de Visual Basic para realizar conversiones implícitas. Las conversiones implícitas pueden producir resultados imprevistos. Por ejemplo, cuando se juntan dos cadenas, Visual Basic concatena las cadenas con independencia de su contenido; por tanto “100” + “10” = “10010.” Si se requiere una conversión estrecha, los datos pueden truncarse. Además, cuando Visual Basic debe realizar conversiones implícitas, el código se ejecuta más lentamente debido al trabajo extra que Visual Basic debe llevar a cabo.

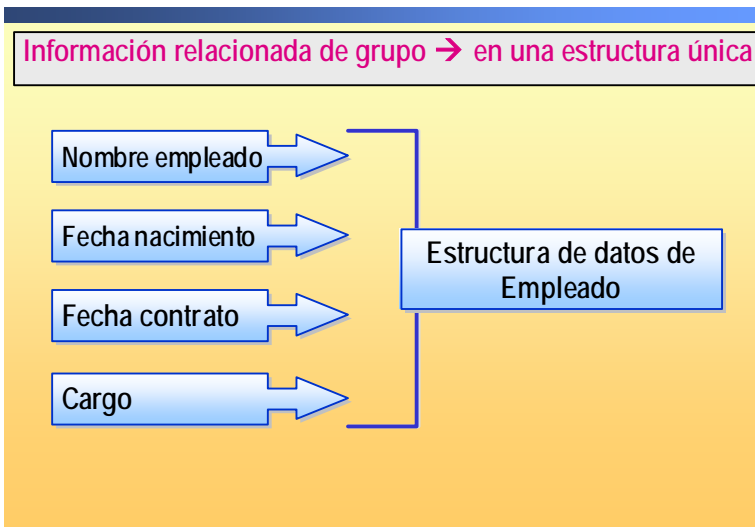
### Cómo afecta Option Strict a las conversiones de datos

Con **Option Strict** deshabilitado (**Off**) (predeterminado), todas las conversiones anchas y estrechas pueden realizarse implícitamente. Con **Option Strict** habilitado (**On**), únicamente están permitidas implícitamente las conversiones anchas, y las conversiones estrechas deben ser explícitas o el compilador generará un error.

---

**Nota** La documentación de Visual Basic .NET incluye más información sobre el parámetro del compilador **Option Strict**.

## Lección: Crear y utilizar estructuras



\*\*\*\*\*

### Introducción

En Visual Basic .NET, podemos combinar variables de varios tipos de datos diferentes para crear un tipo definido por el usuario denominado estructura. Las estructuras resultan útiles cuando deseamos crear una única variable que agrupe varias piezas de información relacionadas. Esta lección explica cómo crear y utilizar estructuras.

### Estructura de la lección

Esta lección incluye los siguientes temas y actividades:

- ¿Qué son las estructuras?
- Cómo declarar estructuras
- Cómo utilizar estructuras
- Práctica: Crear y utilizar estructuras

### Objetivos de la lección

En esta lección, aprenderá a:

- Describir una estructura y sus elementos.
- Declarar y utilizar estructuras.

## ¿Qué son las estructuras?

- Una combinación de tipos de datos
- Se utilizan para crear tipos de valores definidos por el usuario
- Sus miembros pueden ser variables, propiedades, métodos o eventos
- Ejemplo de estructura definida por el usuario:

```
Public Structure Empleado
    Public Nombre As String
    Public Apellido As String
    Public FechaContrato As Date
    Public Cargo As String
    Private Salario As Decimal
End Structure
```

- Ejemplos de estructuras predefinidas: *Point*, *Size*, *Color*

\*\*\*\*\*

### Introducción

Podemos combinar elementos de datos de distintos tipos para crear una combinación de tipos de datos única denominada estructura. Las estructuras resultan útiles cuando deseamos que una única variable guarde varias piezas de información relacionadas. Después de declarar una estructura, podemos declarar variables de ese tipo.

### Definición

Una *estructura* es una combinación de tipos de datos que se crea combinando otros tipos de datos. Las estructuras son de tipo valor (es decir, una variable de tipo estructura contiene los datos de la estructura, en lugar de una referencia a los datos como hace el tipo referencia). Las estructuras pueden tener datos, propiedades, métodos y procedimientos y pueden invocar y manipular eventos.

### Ejemplo

El uso más simple y habitual de las estructuras es encapsular variables relacionadas, creando un tipo de datos definido por el usuario. Por ejemplo, es posible que deseemos guardar juntos el nombre, fecha de contratación, cargo y salario de un empleado. Podríamos utilizar varias variables para esta información, o podemos definir una estructura y utilizarla como la variable de un único empleado. La ventaja de la estructura se hace patente cuando tiene muchos empleados y, por tanto, muchas instancias de la variable. El siguiente ejemplo muestra una estructura *Employee* simple:

```
Public Structure Employee
    Public FirstName As String
    Public LastName As String
    Public HireDate As Date
    Public JobTitle As String
    Private Salary As Decimal
End Structure
```

---

**Nota** El ejemplo anterior utiliza la estructura *Employee* para gestionar una sencilla lista de atributos. La estructura *Employee* también podría tener miembros como un método **Hire** y un evento **Promoted**. Sin embargo, si el diseño de su aplicación puede beneficiarse de utilizar *Employee* como plantilla para otro elemento de programación, como un *Manager* o un *StaffMember*, sería mejor crear *Employee* como clase. Las clases pueden utilizarse como plantillas, y son más extensibles que las estructuras.

---

#### Estructuras proporcionadas en Visual Basic .NET

Visual Basic .NET también proporciona numerosas estructuras predefinidas que podemos utilizar. Por ejemplo, la siguiente lista describe varias estructuras existentes que soportan gráficos en la plataforma .NET. Estas estructuras forman parte del espacio de nombres **System.Drawing**.

- La estructura *Point* representa un par ordenado de coordenadas x e y de enteros que define un punto en un plano bidimensional. Podemos utilizar la estructura *Point* para capturar la ubicación del ratón cuando el usuario hace clic en nuestra aplicación.
- La estructura *Size* representa el tamaño de una región rectangular utilizando un par ordenado de anchura y altura. Utilizamos la estructura *Size* cuando modificamos la propiedad **Size** de un formulario.
- La estructura *Color* representa un color ARGB (*alpha, red, green, blue*). Utilizamos la estructura *Color* cuando modificamos el color de un formulario o de un control.

## Cómo declarar estructuras

- Dentro de un módulo, archivo o clase (no en un procedimiento)
- Sintaxis para declarar estructuras:

```
AccessModifier Structure StructureName
' Declare structure members here
End Structure
```

- Dónde se encuentra el modificador de acceso:
  - **Public** para acceso no restringido
  - **Protected** para acceso sólo dentro de su propia clase
  - **Friend** para acceso en cualquier lugar de la aplicación o ensamblado
  - **Private** para acceso sólo dentro del contexto de su declaración
- No asigne valores a miembros de datos en la declaración

\*\*\*\*\*

Introducción	El primer paso para crear una estructura es declararla. Podemos declarar estructuras en un archivo fuente o dentro de un módulo o clase, pero no dentro de un procedimiento.
Sintaxis	<p>Iniciamos la declaración de una estructura con la instrucción <b>Structure</b>, y la finalizamos con la instrucción <b>End Structure</b>. Entre ambas instrucciones, debemos declarar al menos un miembro. Los miembros pueden ser de cualquier tipo de datos. Podemos utilizar la siguiente sintaxis para declarar una estructura:</p> <pre>AccessModifier Structure StructureName ' Declare structure members here End Structure</pre>
Miembros de la estructura	<p>Debemos declarar cada miembro de los datos de una estructura y especificar un nivel de acceso. Esto significa que cada sentencia de la sección de declaración de variable de la estructura debe utilizar <b>Dim</b>, <b>Friend</b>, <b>Private</b> o <b>Public</b>. Si <b>Option Explicit</b> está <b>On</b>, es una buena idea incluir la cláusula <b>As</b> en cada sentencia. Los miembros declarados con <b>Dim</b> tienen por defecto acceso <b>Public</b>, y los miembros declarados sin la cláusula <b>As</b> tienen por defecto el tipo de datos <b>Object</b>.</p> <p>No podemos inicializar ninguno de los miembros de datos en la declaración de la estructura. Al declarar una variable de tipo estructura, asignamos valores a los miembros accediendo a ellos a través de la variable.</p>



**Niveles de acceso**

Puede accederse a las estructuras desde cualquier lugar del archivo, módulo o clase en el que estén declaradas. Una estructura en un archivo es **Friend** por defecto. Podemos especificar el nivel de acceso de una estructura utilizando el modificador de acceso **Public**, **Protected**, **Friend** o **Private** como se describe en la siguiente tabla:

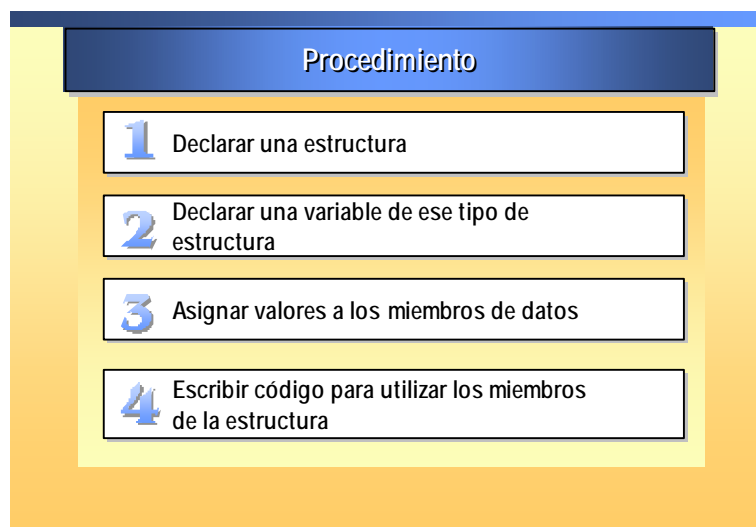
Modificador de acceso	Nivel de acceso
<b>Public</b>	Accesible desde cualquier lugar de la solución o desde cualquier lugar que haga referencia al proyecto
<b>Protected</b>	Accesible únicamente desde dentro de su propia clase
<b>Friend</b>	Accesible desde la aplicación en la que está declarada o desde cualquier otro lugar del ensamblado
<b>Private</b>	Accesible únicamente desde el módulo, archivo u otro elemento de programación en el que esté declarada, incluyendo desde miembros de cualquier tipo anidado, como procedimientos

**Ejemplo**

El siguiente ejemplo muestra cómo declarar una estructura *Employee* para definir un conjunto de datos relacionados con un empleado. Muestra cómo utilizar los modificadores de acceso **Public**, **Friend** y **Private** para reflejar la sensibilidad de los elementos de datos.

```
Public Structure Employee
    ' Public members, accessible throughout declaration region
    Public FirstName As String
    Public MiddleName As String
    Public LastName As String
    ' Friend members, accessible anywhere in the same assembly
    Friend EmployeeNumber As Integer
    Friend BusinessPhone As Long
    ' Private members, accessible only in the structure itself
    Private HomePhone As Long
    Private Salary As Double
    Private Bonus As Double
End Structure
```

## Cómo utilizar las estructuras



\*\*\*\*\*

### Introducción

Una vez creada una estructura, podemos declarar variables a nivel de procedimiento y a nivel de módulo como ese tipo estructura. Podemos asignar valores a los miembros de datos de la estructura de la variable y escribir código para utilizar los miembros de la estructura.

### Procedimiento

Generalmente, los pasos para utilizar las estructuras son los siguientes:

#### 1. Declarar una estructura.

Por ejemplo, podemos crear una estructura que registre información sobre el sistema de un ordenador, como sigue:

```
Private Structure computerInfo
    Public processor As String
    Public memory As Long
    Public purchaseDate As Date
End Structure
```

#### 2. Declarar una variable del tipo de estructura declarado.

Una vez declarada una estructura, podemos crear una variable de ese tipo de estructura, como sigue:

```
Dim mySystem As computerInfo
```

---

3. Asignar valores a los miembros de datos.

Para asignar y recuperar valores desde los elementos de una variable estructura, utilizamos el nombre de la variable que hemos creado junto con el elemento en el bloque de estructura. Separamos el nombre del elemento con un punto. Por ejemplo, podemos acceder e inicializar las variables declaradas en la estructura *computerInfo* como sigue:

```
mySystem.processor = "x86"  
mySystem.purchaseDate = #1/1/2003#  
mySystem.memory = TextBox1.Text
```

4. Escribir código para utilizar los miembros de la estructura.

Después de asignar valores a los miembros de datos, podemos escribir código para utilizarlos desde cualquier lugar de nuestro código que se encuentre dentro de su ámbito. En este ejemplo, los miembros de datos únicamente son accesibles desde dentro del archivo donde la estructura se ha declarado. Por ejemplo, puede verificar si el ordenador tiene suficiente memoria para instalar una determinada aplicación:

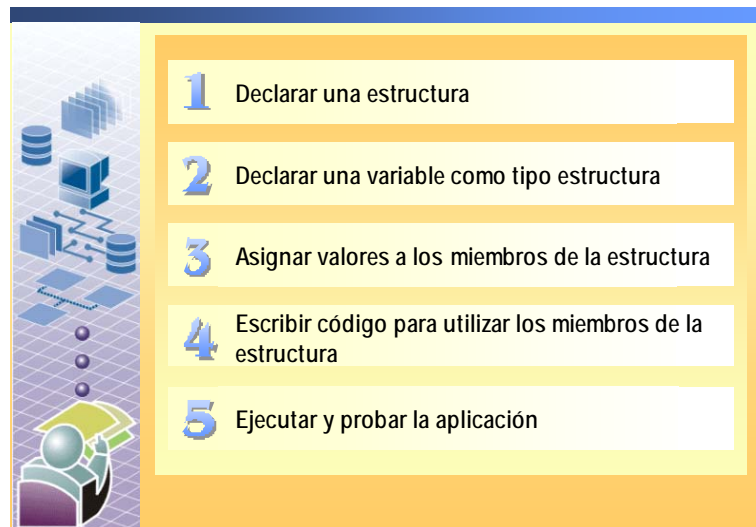
```
If mySystem.memory < 64000 Then NotEnoughMemory = True
```

---

**Nota** Una estructura puede contener cualquiera de los tipos de datos definidos en Visual Basic .NET, incluyendo matrices, objetos y otras estructuras. Encontrará más información sobre la creación de estructuras en “Estructuras y otros elementos de programación” en la documentación de Visual Basic .NET.

---

## Práctica: Crear y utilizar estructuras



\*\*\*\*\*

### Introducción

En esta práctica, crearemos y utilizaremos la estructura y probaremos la aplicación en la que se utiliza la estructura para asegurarnos de que funciona.

### Escenario

Piense en un escenario en el que una aplicación necesite manipular datos sobre una flota de coches para gestionar información sobre el alquiler y mantener el registro de mantenimiento. Seguramente querrá encapsular los detalles de cada coche en una estructura que los desarrolladores puedan utilizar para crear las distintas partes de la aplicación. Limitaremos los detalles relacionados a la marca del vehículo, modelo, precio de compra y fecha de compra.

### Instrucciones

#### 🔗 Abrir el iniciador de código para la práctica

1. Abra Visual Studio .NET.
2. Abra el archivo Structures.sln file, que se encuentra en la carpeta Structure\Starter dentro del archivo practs03.zip.

#### 🔗 Crear una estructura para almacenar información sobre coches

- Declare una estructura pública con el nombre *CarInfo* en la sección de declaraciones de Form1. Incluya los miembros que se muestran en la siguiente tabla.

Nombre del miembro	Modificador de acceso	Tipo de datos
<i>Marca</i>	<b>Dim</b>	<b>String</b>
<i>Modelo</i>	<b>Dim</b>	<b>String</b>
<i>PrecioCompra</i>	<b>Dim</b>	<b>Single</b>
<i>FechaCompra</i>	<b>Dim</b>	<b>Date</b>

### ⚡ Declare una variable de tipo *CarInfo* y asígnele valores

1. En el evento **Button1\_Click**, declarar una variable con el nombre *miCoche* de tipo de estructura *CarInfo*. El código debería ser parecido a:

```
Dim miCoche As CarInfo
```

2. Asignar valores a los miembros de la estructura. Podemos utilizar los siguientes valores o escoger otros.

```
miCoche.Marca = "Jeep"  
miCoche.Modelo = "Cherokee"  
miCoche.PrecioCompra = 27999  
miCoche.FechaCompra = #06/23/2000#
```

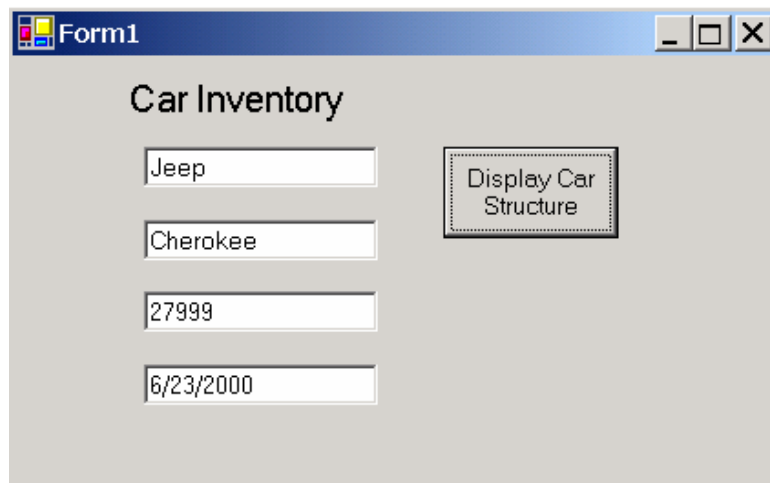
### ⚡ Visualizar la información del coche en los cuadros de texto

- Establezca las propiedades **TextBox** para visualizar la estructura *CarInfo* estructura en cuadros de texto en el formulario. El código debería ser parecido a:

```
TextBox1.Text = miCoche.Marca  
TextBox2.Text = miCoche.Modelo  
TextBox3.Text = miCoche.PrecioCompra  
TextBox4.Text = miCoche.FechaCompra
```

### ⚡ Ejecutar y probar la aplicación

- Ejecutar la aplicación y hacer clic en **Display Car Structure**. El formulario que se ejecute debería tener un aspecto parecido a éste:

The image shows a screenshot of a Windows application window titled "Form1". Inside the window, there is a form titled "Car Inventory". The form contains four text boxes arranged vertically on the left side, each containing a value: "Jeep", "Cherokee", "27999", and "6/23/2000". To the right of these text boxes, there is a button with the text "Display Car Structure". The window has standard Windows controls (minimize, maximize, close) in the top right corner.

#### Archivos de solución

Los archivos de solución de esta práctica se encuentran en la carpeta Structure\Solution dentro del archivo practs03.zip.

## Lección: Almacenar datos en matrices

- ¿Qué es una matriz?
- Cómo declarar una matriz unidimensional
- Cómo utilizar matrices multidimensionales
- Cómo cambiar el tamaño de las matrices

\*\*\*\*\*

**Introducción** Con el uso de matrices, podemos almacenar un grupo de elementos que tienen el mismo tipo de datos bajo un nombre de variable. Utilizamos un número (o *índice*) para hacer referencia a un elemento específico de la serie. Con el uso de matrices, podemos acceder, utilizar y almacenar fácilmente valores del mismo tipo. Esta lección explica cómo declarar y utilizar matrices.

**Estructura de la lección** Esta lección incluye los siguientes temas:

- ¿Qué es una matriz?
- Cómo declarar una matriz unidimensional
- Cómo utilizar matrices multidimensionales
- Cómo cambiar el tamaño de las matrices

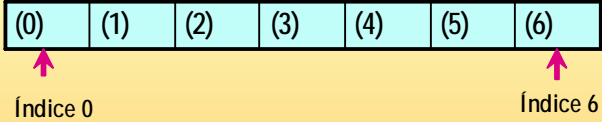
**Objetivos de la lección** En esta lección, aprenderá a:

- Describir la estructura básica de una matriz.
- Declarar e inicializar matrices.
- Cambiar el tamaño de matrices.
- Utilizar matrices multidimensionales.

## ¿Qué es una matriz?

■ **Definición:** Una matriz es una serie de elementos de datos

- Todos los elementos de una matriz tienen el mismo tipo de datos
- Se accede a los elementos individuales utilizando índices enteros



El diagrama muestra una fila de siete cuadros rectangulares, cada uno con un número entre paréntesis: (0), (1), (2), (3), (4), (5) y (6). Debajo del primer cuadro (0) hay una flecha roja que apunta hacia él y el texto 'Índice 0'. Debajo del último cuadro (6) hay una flecha roja que apunta hacia él y el texto 'Índice 6'.

■ **Ejemplo**

- Para declarar una matriz entera con siete elementos:

```
Dim countHouses(6) As Integer
```

- Para acceder al tercer elemento de la matriz:

```
TextBox1.Text = CStr(countHouses(2))
```

\*\*\*\*\*

### Introducción

Una matriz es un tipo de datos fundamental en Visual Basic, al igual que en la mayoría de lenguajes de programación. Podemos utilizar matrices para crear menos código y más sencillo en muchas situaciones porque podemos utilizar estructuras y bucles de decisión para recuperar y modificar cualquier elemento de una matriz.

### Definiciones

Una *matriz* es una secuencia de elementos de datos del mismo tipo. Podemos acceder a elementos individuales de una matriz utilizando el nombre de la matriz y un índice o índices (empezando por 0) para especificar la posición del elemento en la matriz. Una matriz tiene una o más dimensiones con uno o más elementos en cada dimensión.

### Analogía

Piense en un vecindario con una calle y siete casas como analogía para una matriz unidimensional. Piense en el vecindario como matriz, la calle como dimensión de la matriz, y las casas como elementos individuales de la matriz. El tipo de datos de los elementos de la matriz podría ser **Integer**.

### Ejemplo

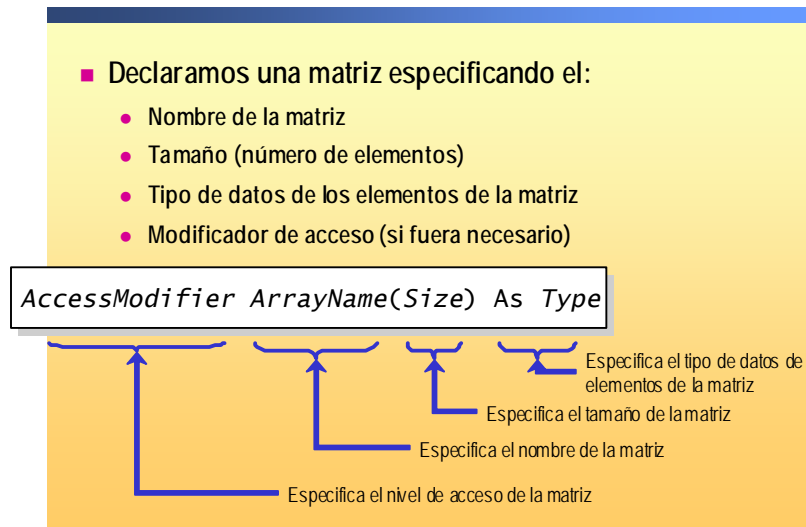
Para declarar la matriz descrita en la analogía anterior, podemos utilizar el código siguiente:

```
Dim countHouses(6) As Integer
```

Podemos acceder a la tercera casa de la matriz *countHouses* como sigue:

```
TextBox1.Text = CStr(countHouses(2))
```

## Cómo declarar una matriz unidimensional



\*\*\*\*\*

**Introducción** Declaramos una matriz unidimensional especificando su nombre, tamaño, tipo de datos de los elementos y su nivel de acceso.

**Sintaxis** Creamos o declaramos matrices en el código del mismo modo que declaramos otras variables. Debemos seguir las mismas directrices para establecer el nombre, ámbito y seleccionar los tipos de datos. Para declarar matrices, utilizar la siguiente sintaxis:

*AccessModifier ArrayName(Size) As Type*

El argumento **Size** especifica el número inicial de elementos de la matriz. En ocasiones, no es necesario especificar un modificador de acceso debido a que el ámbito de la matriz está determinado por su ubicación en el código. En ese caso, declare la matriz con la palabra clave **Dim**.

**Ejemplos de matrices locales**

Podemos declarar matrices locales utilizando la instrucción **Dim** e insertando la declaración dentro de un procedimiento, como se muestra en el siguiente ejemplo:

```
Sub InitializeArray( )
    ' Allocate 31 elements (0) to (30)
    Dim currentExpense(30) As Decimal
    Dim number As Integer
    For number = 0 to 30
        currentExpense(number) = 100
    Next number
End Sub
```

Este ejemplo almacena el gasto diario de cada día del mes en una matriz de 31 elementos denominada *currentExpense*. Cada elemento de la matriz contiene un valor, y accedemos a los valores especificando el índice del elemento. Cada elemento de la matriz tiene asignado un valor de 100.



### Ejemplos de matrices públicas

Para crear una matriz pública, utilizaremos la palabra clave **Public** en lugar de **Dim**, como se muestra en el siguiente ejemplo:

```
Public Counters(14) As Integer  
Public Sums(20) As Double
```

La primera declaración crea una matriz unidimensional con 15 elementos, con números de índice de 0 a 14. La segunda declaración crea una matriz con 21 elementos, con números de índice de 0 a 20.

Podemos asignar valores a los elementos de la matriz después de declararla, o inicializarla cuando la declaremos, encerrando los valores entre los corchetes, como se muestra en el siguiente ejemplo:

```
Public Counters( ) As Integer = {1, 2, 3, 4, 5, 6, 7}
```

## Cómo utilizar matrices multidimensionales

- Especificar todas las dimensiones y elementos
- Total elementos = producto de todos los tamaños
- Declarar una variable de matriz multidimensional :
  - Añadir un par de paréntesis tras el nombre de la variable
  - Colocar comas dentro de los paréntesis para separar las dimensiones
  - Iniciar la declaración con la sentencia **Dim** o un modificador de acceso
- Ejemplo:

```
Public ThreeDimensions(3,9,14) As Double
' Three-dimensional array
```

\*\*\*\*\*

### Introducción

Una matriz puede tener más de una dimensión. La *dimensionalidad*, o *rango*, corresponde al número de índices utilizados para identificar un elemento individual de la matriz. Podemos especificar hasta 32 dimensiones para una matriz; sin embargo, rara vez necesitaremos más de tres dimensiones.

### Declarar variables en una matriz multidimensional

Para declarar una variable de una matriz multidimensional, añada un par de paréntesis detrás del nombre de la variable y coloque comas dentro del paréntesis para separar las dimensiones, como en el siguiente ejemplo:

```
' Four-dimensional array
Dim My4DArray(2, 6, 4, 1) As Integer
```

Como palabra clave, utilice **Dim** para una matriz local, o especifique un modificador de acceso como **Public** o **Friend** para proporcionar un nivel de ámbito distinto.

### Ejemplo de una matriz de dos dimensiones

Piense en una matriz de dos dimensiones como si fuese una cuadrícula. El siguiente ejemplo muestra cómo declarar una matriz de dos dimensiones con 4 filas y 3 columnas.

```
Dim storageNumber(3, 2) As Double
```

Para asignar un valor a un elemento específico de la matriz, haga referencia a los números de índice del elemento. Por ejemplo, el siguiente código muestra cómo asignar un valor a un elemento específico de la matriz declarado en el código anterior.

```
storageNumber(2, 1) = 24
```

La siguiente ilustración es una representación de la matriz de dos dimensiones creada en este ejemplo:

		Column Indexes		
Row Indexes		0	1	2
	0	12	22	32
	1	13	23	33
	2	14	24	34
	3	15	25	35

storageNumber(2, 1) = 24

Ejemplo de una matriz de tres dimensiones

El siguiente ejemplo muestra cómo crear y acceder a una matriz pública de tres dimensiones:

```
Public ThreeDimensions(3, 9, 14) As String
```

Esta declaración crea una matriz con tres dimensiones de tamaños 4, 10 y 15. El número total de elementos es el producto de estas tres dimensiones, 600. Imagine una matriz de tres dimensiones como si fuese un cubo.

El siguiente código muestra cómo asignar el valor almacenado en un elemento específico de la matriz anterior a la propiedad **Text** de un cuadro de texto.

```
TextBox1.Text = ThreeDimensions(2,6,4)
```

---

**Nota** Cuando añadimos dimensiones a una matriz, el almacenamiento total que necesita la matriz aumenta enormemente. Por ello, debemos evitar declarar matrices más grandes de lo necesario.

---

## Cómo cambiar el tamaño de una matriz

- Podemos cambiar el tamaño de una matriz en cualquier momento
- Utilizar la instrucción ReDim
- Sintaxis para cambiar el tamaño de una matriz:

```
ReDim matrizExistente(NuevoTamaño)
```

- Ejemplo:

```
Dim miMatriz(,) ' Declare array
ReDim miMatriz(3, 5) ' Redimension array
```

\*\*\*\*\*

### Introducción

En Visual Basic .NET, podemos cambiar el tamaño de una matriz en cualquier momento especificando la nueva dimensión. Podemos, incluso, cambiar el número de dimensiones de la matriz. Modificar el tamaño de matrices nos ayuda a gestionar la memoria eficazmente. Por ejemplo, podemos utilizar una matriz grande durante un corto periodo de tiempo y cambiar su tamaño por otro más pequeño. De esta forma, se libera memoria que ya no necesitamos.

### Sintaxis

Utilizar la siguiente sintaxis para cambiar el tamaño de una matriz unidimensional existente:

```
ReDim matrizExistente(NuevoTamaño)
```

Si no estamos seguros del tamaño de matriz que necesitamos, podemos declarar la matriz sin especificar el tamaño de las dimensiones, como se muestra en la ilustración anterior.

### Ejemplos

El siguiente ejemplo muestra cómo utilizar la instrucción **ReDim** para asignar y reasignar espacio de almacenamiento para variables de matrices.

```
Dim number, miMatriz( ) As Integer
' Declare variable and array variable
' Allocate 6 elements
ReDim miMatriz(5)
For number = 0 to 5
' Initialize array
miMatriz(number) = number
Next number
```

La siguiente instrucción cambia el tamaño de la matriz sin guardar el contenido de los elementos.

```
' Resize to 11 elements
ReDim miMatriz(10)
For number = 0 To 10
' Initialize array
    miMatriz(number) = number
Next number
```

Conservar los valores  
originales de la matriz

Cuando utilizamos la instrucción **ReDim** en una matriz, normalmente sus valores existentes se pierden. No obstante, podemos conservarlos incluyendo la palabra clave **Preserve** en la instrucción **ReDim**. Por ejemplo, la siguiente instrucción asigna una nueva matriz, inicializa sus elementos desde los elementos correspondientes de la matriz existente (**miMatriz**), y asigna la nueva matriz a **miMatriz**.

```
ReDim Preserve miMatriz(10)
```