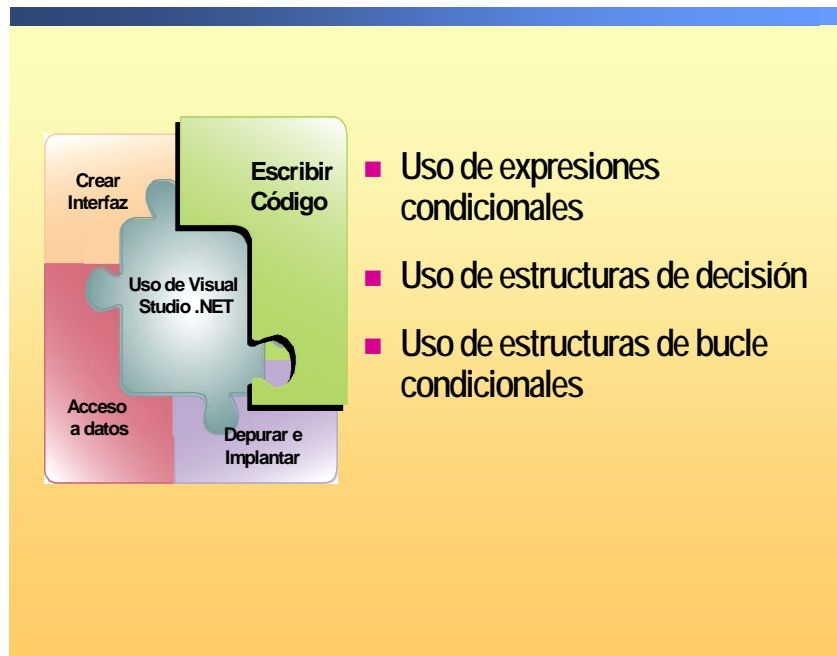


Bucles y estructuras de decisión

Índice

Descripción	1
Lección: Uso de expresiones condicionales	2
Lección: Uso de estructuras de decisión	14
Lección: Uso de estructuras de bucle condicionales	21

Descripción



Introducción

El funcionamiento principal de las aplicaciones basadas en Microsoft® Visual Basic® consiste en crear procedimientos que se ejecutan cuando sucede un determinado evento. Por ejemplo, escribimos código para un procedimiento **Button_Click** que se ejecuta cuando un usuario hace clic en el botón.

Podemos extender este modelo de programación basada en eventos utilizando estructuras de decisión y bucles. Utilizamos estructuras de decisión para comparar valores y, a continuación, ejecutar una determinada sección de código en base al resultado. Utilizamos estructuras de bucle para ejecutar una serie de instrucciones repetidamente hasta que se satisface una condición. Este módulo explica cómo implementar estructuras de decisión y estructuras de bucle para controlar la salida y la ejecución de un programa.

Objetivos

En este módulo, aprenderemos a:

- Crear fórmulas y expresiones condicionales utilizando operadores aritméticos, de comparación y lógicos.
- Utilizar instrucciones **If...Then** para evaluar si una condición es verdadera o falsa y dirigir el flujo del programa en consecuencia.
- Utilizar instrucciones **Select Case** para probar diferentes valores de la misma expresión y ejecutar las instrucciones correspondientes.
- Utilizar instrucciones **Do...Loop** para ejecutar instrucciones hasta o mientras se satisface una condición específica.
- Utilizar instrucciones **For...Next** para ejecutar instrucciones un determinado número de veces.
- Escoger la estructura de decisión o bucle adecuado en función de los requerimientos de la aplicación.

Lección: uso de expresiones condicionales

- ¿Qué son las expresiones condicionales?
- Cómo utilizar operadores aritméticos
- Cómo utilizar operadores de comparación
- Cómo utilizar operadores lógicos
- Cómo combinar operadores lógicos y de comparación

Cuando desarrollamos aplicaciones en Microsoft Visual Basic .NET, necesitamos escribir expresiones que el equipo pueda evaluar. Las expresiones que evalúan **True** o **False** se denominan *expresiones condicionales*. Esta lección explica cómo construir expresiones condicionales utilizando operadores básicos aritméticos, de comparación y lógicos.

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

- ¿Qué son las expresiones condicionales?
- Cómo utilizar operadores aritméticos
- Cómo utilizar operadores de comparación
- Cómo utilizar operadores lógicos
- Cómo combinar operadores de comparación y lógicos
- Práctica: evaluar expresiones condicionales

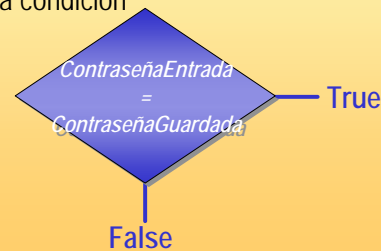
Objetivos de la lección En esta lección, aprenderemos a:

- Describir operadores matemáticos, lógicos y de comparación.
- Utilizar operadores aritméticos, lógicos y de comparación para crear fórmulas y evaluar condiciones.

¿Qué son las expresiones condicionales?

■ Expresiones condicionales:

- Incluyen una condición que debe evaluarse si es **True** o **False**
- Incluyen un operador para especificar cual es el resultado de la condición



Si la contraseña es la correcta, la condición es True

Las expresiones condicionales permiten desarrollar la aplicación para realizar tareas de forma selectiva, en función de valores generados por la aplicación o de valores introducidos por el usuario. Las expresiones condicionales tienen numerosos usos en programación, pero generalmente se encuentran en instrucciones que controlan la ejecución de programas.

Condiciones de la prueba

Las expresiones condicionales deben devolver **True** o **False**. La expresión puede ser tan simple como una variable definida de tipo Boolean (o un tipo de variable que pueda ser convertida implícitamente en Boolean), o puede ser una combinación de variables y literales unidos por un operador que devuelve un resultado Boolean.

Ejemplos de expresiones condicionales

La siguiente tabla ofrece una lista de ejemplos de expresiones condicionales y de los valores que devuelven.

Expresión condicional	Valor que devuelve la expresión
$5 > 2$	True.
DateToday = #11/30/99#	False el día que leemos esta fecha.
ContraseñaEntrada = ContraseñaGuardada	True si <i>ContraseñaEntrada</i> es la misma que <i>ContraseñaGuardada</i> .
MyFirstName = "John"	True si nuestro nombre es John. False si tenemos otro nombre.

Cómo utilizar operadores aritméticos

- Símbolos que evalúan expresiones condicionales
- Pueden realizar operaciones aritméticas
- Sintaxis:

```
expression1 arithmetic operator expression2
```

- Ejemplo:

```
Dim x As Integer
x = 52 * 17
x = 120 / 4
x = 67 + 34
x = 32 - 12
x = 23 ^ 3
```

Los operadores aritméticos se utilizan para realizar muchas de las operaciones aritméticas familiares que implican el cálculo de valores numéricos representados por literales, variables, otras expresiones, llamadas de funciones y propiedades, y constantes.

Operadores aritméticos

La siguiente tabla presenta una lista de los operadores aritméticos que utilizaremos con mayor frecuencia en Visual Basic .NET.

Operador aritmético	Descripción
*	Multiplicación
/	División
+	Suma
-	Resta
^	Exponenciación

Sintaxis

La sintaxis general para operadores aritméticos es la siguiente:

```
expression1 arithmetic operator expression2
```

Ejemplo de utilización de operadores aritméticos

El siguiente ejemplo muestra los operadores aritméticos que se utilizan más habitualmente:

```
Dim x As Integer
x = 52 * 17
x = 120 / 4
x = 67 + 34
x = 32 - 12
x = 23 ^ 3
```

Cómo utilizar operadores de comparación

- Símbolos que evalúan expresiones condicionales y devuelven un valor Boolean
- Pueden comparar números o cadenas
- Sintaxis:

```
expression1 comparison operator expression2
```

- Ejemplo:

```
Dim Quantity As Integer
Dim LargeOrder As Boolean
LargeOrder = Quantity > 1000
```

Un operador de comparación realiza una operación sobre dos valores y devuelve un resultado Boolean. Visual Basic .NET soporta seis operadores de comparación.

Valores Boolean

Cuando Visual Basic .NET compara dos expresiones, devuelve uno de los dos valores siguientes: **True** o **False**. Estos valores se denominan *valores Boolean*. Cuando los valores Boolean se convierten en tipos numéricos, **False** pasa a ser 0 y **True** es -1. Los operadores de comparación devuelven uno de estos valores Boolean cuando prueban una expresión condicional.

Operadores de comparación

Los seis operadores de comparación que podemos utilizar en Visual Basic .NET son los siguientes:

Operador	Descripción
<	Menor que
<=	Menor o igual que
>	Mayor que
>=	Mayor o igual que
=	Igual que
<>	Distinto de

Sintaxis

La sintaxis general para los operadores de comparación es la siguiente:

```
expression1 comparison operator expression2
```

Ejemplo de utilización de operadores de comparación

Por ejemplo, su aplicación de procesamiento de órdenes de venta proporcionará un descuento a los pedidos en los cuales la cantidad del artículo solicitado sea superior a 1.000. El siguiente código muestra cómo podemos referenciar esta prueba más de una vez, por ello, decidimos guardar el resultado de la prueba en una variable Boolean denominada *LargeOrder*.

```
Dim Quantity As Integer
Dim LargeOrder As Boolean
LargeOrder = Quantity > 1000
```

Ejemplo de comparación de cadenas

Cuando Visual Basic .NET compara las dos cadenas, compara cada cadena carácter a carácter hasta que se encuentra la primera discordancia. Dos cadenas son iguales únicamente si ambas cadenas tienen la misma longitud y contienen exactamente la misma secuencia de caracteres. Cada carácter tiene un valor numérico para las comparaciones.

Expresiones de cadenas	Resultado
"DOG" = "DOG"	True. Ambas cadenas son idénticas.
"DOG" = "dog"	False. Aunque ambas cadenas contienen <i>dog</i> , una cadena está en mayúsculas y la otra en minúsculas.

Cómo utilizar los operadores lógicos

- Los operadores lógicos realizan una evaluación lógica de expresiones y devuelven un valor Boolean

- Sintaxis:

```
expression1 logical operator expression2
```

- Ejemplo:

```
OrderedLastMonth And OrderDelivered
```

Utilizamos operadores lógicos para evaluar una o más expresiones condicionales y devolver un único valor Boolean **True** o **False**. En Visual Basic .NET, podemos utilizar seis operadores lógicos.

Operadores lógicos

La siguiente tabla muestra los seis operadores lógicos utilizados en Visual Basic .NET. Observe que todos los operadores excepto **Not** trabajan con dos expresiones, que también se denominan *operandos*.

Operador	Función
And	Combina dos expresiones. Cada expresión debe ser True para que toda la expresión sea True . True And True = True False And False = False True And False = False False And True = False
Or	Combina dos expresiones. Si una expresión es True , toda la expresión es True . True Or True = True True Or False = True False Or True = True False Or False = False
Not	Proporciona el negativo lógico de la entrada. Not(True) = False Not(False) = True
AndAlso	Si la primera expresión de una expresión AndAlso es False , la segunda expresión no se evalúa y se devuelve False para la expresión AndAlso .

(continuación)

Operador	Función
OrElse	Si la primera expresión de una expresión OrElse es True , la segunda expresión no se evalúa y se devuelve True para la expresión OrElse .
Xor	Combina dos expresiones. Toda la expresión se considera True si las dos expresiones no son ambas True ni ambas False . True Xor True = False True Xor False = True False Xor True = True False Xor False = False

Sintaxis La sintaxis general para operadores lógicos es la siguiente:

```
expression1 logical operator expression2
```

Ejemplos de operadores lógicos La siguiente tabla proporciona ejemplos de los operadores lógicos **And** y **Or**.

Ejemplo de código	Resultado
OrderedLastMonth And OrderDelivered	Devuelve True si el último pedido se realizó en el mes anterior y ha sido entregado. <i>OrderedLastMonth</i> y <i>OrderDelivered</i> son variables Boolean.
ChildMembership Or AdultMembership	Devuelve True si el miembro es un niño o un adulto.

Cómo combinar operadores lógicos y de comparación

- Podemos combinar operadores de comparación y operadores lógicos con instrucciones condicionales

- Ejemplo:

Operadores de comparación

Operador lógico

```
LateActiveCustomer = DaysOverDue >= 60 And ActiveCustomer
```

Es posible que necesitemos combinar operadores de comparación y operadores lógicos en nuestras expresiones condicionales. Cuando se combinan operadores, es necesario saber el orden en que Visual Basic .NET resuelve los operadores. Visual Basic .NET resuelve múltiples operaciones en una expresión siguiendo un conjunto de reglas denominado *prioridad de operadores*.

Ejemplo de combinación de operadores

El siguiente ejemplo combina operadores de comparación con un operador lógico:

```
LateActiveCustomer = DaysOverDue >= 60 And ActiveCustomer
```

Prioridad de operadores

Cuando se combinan operadores lógicos y de comparación en una misma sentencia de código, los operadores de comparación se evalúan después de los operadores aritméticos y de concatenación pero antes de los operadores lógicos.

El resultado del siguiente ejemplo es 130 ya que la operación de multiplicación tiene prioridad:

$$x = 100 + 10 * 3$$

Los operadores lógicos se evalúan en el orden siguiente:

1. **Not**
2. **And**
3. **Or**
4. **Xor**

Si se utiliza el mismo operador lógico o de comparación varias veces en una misma línea de código, los operadores se evalúan de izquierda a derecha.

Imponer el orden de evaluación de los operadores

La prioridad de operadores puede ser a menudo fuente de errores lógicos en un programa. Podemos incluir paréntesis para imponer el orden de evaluación. Por ejemplo, la tabla lógica de la siguiente expresión se muestra debajo:

Respuesta = A And B Or C And D


A	B	C	D	Respuesta
False	False	False	False	False
False	False	False	True	False
False	False	True	False	False
False	False	True	True	True
False	True	False	False	False
False	True	False	True	False
False	True	True	False	False
False	True	True	True	True
True	False	False	False	False
True	False	False	True	False
True	False	True	False	False
True	False	True	True	True
True	True	False	False	True
True	True	False	True	True
True	True	True	False	True
True	True	True	True	True

Ahora examine la tabla lógica si cambiamos el orden de evaluación. Las respuestas diferentes se muestran en negrita:

Respuesta = A And (B Or C) And D

A	B	C	D	Respuesta
False	False	False	False	False
False	False	False	True	False
False	False	True	False	False
False	False	True	True	False
False	True	False	False	False
False	True	False	True	False
False	True	True	False	False
False	True	True	True	False
True	False	False	False	False
True	False	False	True	False
True	False	True	False	False
True	False	True	True	True
True	True	False	False	False
True	True	False	True	True
True	True	True	False	False
True	True	True	True	True

Práctica: Evaluar expresiones condicionales




■ Utilice la aplicación de ejemplo para calcular los resultados de estas expresiones:

TestString = TestString	0 And 0
TestString = Teststring	-1 And 0
TestString < TestString	-1 And -1
Test < TestString	-1 Or -1
100 > 10	-1 Xor -1
10 < 10	-1 Xor 0
10 <= 10	0 Xor 0

🔗 Abrir Microsoft Visual Studio® .NET y cargar el proyecto de ejemplo

- Haga clic en **Inicio**, seleccione **Todos los programas**, seleccione **Microsoft Visual Studio .NET 2003** y haga clic en **Microsoft Visual Studio .NET 2003**.
- En la página de inicio, haga clic en **Inicio** y, a continuación, clic en **Abrir proyecto**.
En el cuadro de diálogo **Abrir proyecto**, vaya a la carpeta ExpressionPractice dentro del archivo practs05.zip, y haga doble clic en **ExpressionPractice.sln**.

🔗 Ejecutar la aplicación

- En la barra de herramientas estándar, haga clic en **Iniciar** .
- Calcule la respuestas para las expresiones que se listan en la siguiente tabla y utilizar la aplicación para comprobar sus cálculos.
 - Para comprobar las siguientes expresiones, pasar los valores como valores **String** dejando sin seleccionar la caja de verificación de **Treat Values As Numeric**.

Expresión	Resultado (escribir el resultado de la aplicación)
TestString = TestString	True (-1)
TestString = Teststring	False (0)
TestString < TestString	False (0)
Test < TestString	True (-1)

- b. Para comprobar las siguientes expresiones, pasar los valores como valores **Numeric** seleccionando **Treat Values As Numeric**. En las expresiones que utilizan operadores lógicos, recuerde que -1 es **True** y 0 es **False**.

Expresión	Resultado (escribir el resultado de la aplicación)
100 > 10	True (-1)
10 < 10	False (0)
10 <= 10	True (-1)
0 And 0	False (0)
-1 And 0	False (0)
-1 And -1	True (-1)
-1 Or -1	True (-1)
-1 Xor -1	False (0)
-1 Xor 0	True (-1)
0 Xor 0	False (0)

⏮ Cerrar de la aplicación

Lección: uso de estructuras de decisión

■ If...Then	■ If...Then...Else
<pre>If Sales > 10000 Then Bonus = .10 * Sales End If</pre>	<pre>If Sales > 10000 Then Bonus = .10 * Sales Else Bonus = 0 End If</pre>
■ If...Then...ElseIf	■ Select Case
<pre>If Sales > 10000 Then Bonus = .10 * Sales ElseIf Sales > 5000 Then Bonus = .05 * Sales Else Bonus = .02 * Sales End If</pre>	<pre>Select Case Rank Case 1 Bonus = 0 Case 2,3 Bonus = .05 * Sales Case 4 to 6 Bonus = .10 * Sales Case Else Bonus = .15 * Sales End Select</pre>

En nuestro programa, podemos desear que algunas secciones de su código se ejecuten únicamente cuando se cumpla una determinada condición. Esto se consigue utilizando una estructura de decisión. Esta lección explica cómo utilizar las estructuras de decisión **If...Then** y **Select Case**. Estudiaremos cómo incorporar expresiones condicionales en estas estructuras.

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

- Cómo utilizar instrucciones **If...Then**
- Cómo utilizar instrucciones **If...Then...Else**
- Cómo utilizar instrucciones **If...Then...ElseIf**
- Cómo utilizar instrucciones **Select Case**
- Directrices para elegir una estructura de decisión

Objetivos de la lección En esta lección, aprenderemos a:

- Utilizar instrucciones **If...Then**.
- Utilizar instrucciones **If...Then...Else**.
- Utilizar instrucciones **If...Then...ElseIf**.
- Utilizar instrucciones **Select Case**.
- Elegir la estructura de decisión adecuada para un determinado escenario.

Cómo utilizar instrucciones If...Then

- Se utilizan para una decisión True o False
- Si la condición es True, se ejecutan las instrucciones que siguen a la instrucción If
- Si la condición es False, las instrucciones que siguen a la instrucción If no se ejecutan

```
If Sales > 10000 Then
    Bonus = .10 * Sales
End If
```

En general, la instrucción **If...Then** se utiliza cuando el programa debe evaluar si una instrucción es verdadera o falsa.

Las instrucciones **If...Then** evalúan si una condición es verdadera o falsa y dirigen el flujo del programa en consecuencia.

Sintaxis

Las instrucciones **If...Then** pueden utilizar sintaxis en una sola línea o en forma de bloque. Observe que la versión en una sola línea no requiere el uso de una instrucción **End If**. Normalmente, la instrucción **If...Then** en una sola línea se utiliza para ejecutar únicamente una instrucción condicionalmente. En instrucciones **If...Then** en forma de bloque, es habitual indentar el código entre las instrucciones **If** y **End If**. La mejor práctica es, sencillamente, facilitar la legibilidad.

Podemos utilizar la siguiente sintaxis en una línea:

```
If condition Then clause
```

También podemos utilizar la siguiente sintaxis en forma de bloque:

```
If condition Then
    statements
End If
```

Ejemplo de una instrucción If...Then

Este ejemplo compara el valor de un entero denominado *Sales* con 10.000. Si *Sales* es superior a 10.000, se calcula un 10 por ciento de *bonus*. Si el valor de *Sales* no supera 10.000, se ignora la línea de código detrás del **Then** y no se calcula el *bonus*.

```
If Sales > 10000 Then
    Bonus = .10 * Sales
End If
```

Cómo utilizar instrucciones If...Then...Else

- Se utilizan para una decisión con dos opciones como mínimo
- Cada instrucción If debe tener un End If correspondiente
- Si la condición es True, se ejecutarán las instrucciones que siguen a la instrucción If
- Si la condición es False, no se ejecutarán las instrucciones que siguen a la instrucción If

```
If Sales > 10000 Then
    Bonus = .10 * Sales
Else
    Bonus = 0
End If
```

Introducción

Las instrucciones **If...Then...Else** son una extensión del concepto **If...Then**. Utilizando un bloque **If...Then...Else**, podemos definir dos bloques de código y que nuestro programa ejecute uno u otro dependiendo del resultado de una condición. Si es verdadera más de una de las condiciones de una estructura condicional, únicamente se ejecutarán las instrucciones de código incluidas en la primera condición verdadera.

Sintaxis

Una instrucción **If...Then...Else** incluye los siguientes componentes:

- Una condición que evalúa si es **True** o **False**
- Una o más instrucciones que se ejecutan dependiendo del resultado de la prueba de la condición
- Una instrucción **End If** en el caso de un bloque

Para utilizar una instrucción **If...Then...Else**, utilizar la siguiente sintaxis:

```
If condition Then
    statements
Else
    statements
End If
```

Ejemplo de una instrucción If...Then...Else

El siguiente ejemplo compara el valor de un entero denominado *Sales* con 10.000. Si *Sales* es superior a 10.000, se calcula un 10 por ciento de *bonus*. Si el valor de *Sales* no supera 10.000, el *bonus* es igual a cero.

```
If Sales > 10000 Then
    Bonus = .10 * Sales
Else
    Bonus = 0
End If
```

**Anidar instrucciones
If...Then**

Podemos utilizar tantas instrucciones **If...Then** como sean necesarias en una estructura **If...Then**. Añadir instrucciones **If...Then** en una estructura **If...Then** se denomina *anidación*.

Ejemplo de anidación

Por ejemplo, el siguiente código muestra cómo implementar un *bonus* de ventas otorgado a un empleado a tiempo parcial menor que el correspondiente a un empleado de jornada completa.

```
If Sales > 10000 Then
  If EmployeeType = "Full-Time" Then
    Bonus = .10 * Sales
  Else
    Bonus = .08 * Sales
  End If
Else
  Bonus = 0
End If
```

Cómo utilizar instrucciones If...Then...ElseIf

- Se utilizan para anidar instrucciones de decisión
- Cada instrucción If debe tener su correspondiente End If
- Las instrucciones ElseIf no tienen su propio End If
- Las instrucciones ElseIf no pueden aparecer después de Else
- Si la condición es True, se ejecutan las instrucciones que siguen a la instrucción If

```
If Sales > 10000 Then
    Bonus = .10 * Sales
ElseIf Sales > 5000 Then
    Bonus = .05 * Sales
Else
    Bonus = .02 * Sales
End If
```

Introducción

Las instrucciones **If...Then...ElseIf** son como las instrucciones **If...Then...Else**, excepto en que permiten que nuestro programa elija entre más de dos alternativas.

Sintaxis

Para utilizar una instrucción **If...Then...ElseIf**, utilizar la siguiente sintaxis:

```
If condition Then
    statements
ElseIf condition2 Then
    statements
Else
    statements
End If
```

Ejemplo de sentencia If...Then...ElseIf

El siguiente ejemplo compara en primer lugar el valor de un entero denominado *Sales* con 10.000, y a continuación lo compara con 5.000. Si *Sales* es superior a 10.000, se calcula un 10 por ciento de *bonus*. Si *Sales* es superior a 5.000, se calcula un 5 por ciento de *bonus*. Si el valor de *Sales* no es superior a 10.000 y tampoco a 5.000, se calcula un 2 por ciento de *bonus*.

```
If Sales > 10000 Then
    Bonus = .10 * Sales
ElseIf Sales > 5000 Then
    Bonus = .05 * Sales
Else
    Bonus = .02 * Sales
End If
```

Cómo utilizar instrucciones Select Case

- Seleccionan un bloque de código a ejecutar basándose en una lista de posibles elecciones
- Se utilizan como alternativa a complejas instrucciones If...Then...Else anidadas
- Si varias instrucciones Case son verdaderas, únicamente se ejecutan las instrucciones que pertenecen a la primera instrucción Case verdadera

```

Select Case Rank
  Case 1
    Bonus = 0
  Case 2,3
    Bonus = .05 * Sales
  Case 4 to 6
    Bonus = .10 * Sales
  Case Else
    Bonus = .15 * Sales
End Select

```

Introducción

La instrucción **Select Case** permite a nuestra aplicación ejecutar uno o varios bloques de código dependiendo del valor de una expresión de prueba. La instrucción **Select Case** funciona como una compleja estructura **If...Then...Else** anidada, pero su mantenimiento es más sencillo.

Sintaxis

La sintaxis para la instrucción **Select Case** es como sigue:

```

Select Case testexpression
  [Case expressionlist-n]
    [statements-n1] . . .
  [Case Else]
    [elstatement]
End Select

```

Ejemplo de una instrucción Select Case

El siguiente ejemplo muestra cómo utilizar la instrucción **Select Case** para evaluar el valor de un *bonus* basado en seis condiciones distintas. Si ninguna de estas condiciones es verdadera, el *bonus* será del 15 por ciento.

```

Dim Rank As Integer
[Set Rank value]
...
Select Case Rank
  Case 1
    Bonus = 0
  Case 2, 3
    Bonus = .05 * Sales
  Case 4 to 6
    Bonus = .10 * Sales
  Case Else
    Bonus = .15 * Sales
End Select

```

Directrices para elegir una estructura de decisión

- Las instrucciones **If...Then** se utilizan para controlar la ejecución de un único bloque de código
- Las instrucciones **If...Then...Else** se utilizan para controlar la ejecución de dos secciones de código mutuamente excluyentes
- Las instrucciones **Select Case** se utilizan cuando se dispone de una lista de valores posibles

Introducción

Debemos escoger cuidadosamente la estructura de decisión a utilizar, ya que Visual Basic .NET saldrá de nuestra estructura de decisión tan pronto como encuentre la primera instrucción verdadera. Esto significa que las condiciones que sigan a la primera instrucción verdadera no serán probadas.

Directrices

Seguir estas directrices para determinar qué estructura de decisión utilizar:

- Utilizar instrucciones **If...Then** para controlar la ejecución de un único bloque de código.
- Utilizar instrucciones **If...Then...Else** para controlar la ejecución de dos secciones mutuamente excluyentes de código. Son posibles más secciones si utiliza instrucciones **ElseIf**. Pero si ha utilizado varias instrucciones **ElseIf**, es posible que desee utilizar una instrucción **Select Case**.
- Utilizar instrucciones **Select Case** cuando tenga una lista de valores posibles.

Lección: Uso de estructuras de bucle condicionales

- Cómo utilizar instrucciones **For...Next**
- Cómo utilizar instrucciones **For Each...Next**
- Cómo utilizar instrucciones **Do...Loop**
- Cómo utilizar instrucciones **Exit**

Introducción

Es probable que necesite frecuentemente repetir la ejecución de un bloque de código hasta que se cumpla una determinada condición en los programas que desarrolle. Para repetir la ejecución de un bloque de código en Visual Basic .NET, es conveniente utilizar estructuras de bucle condicionales. Esta lección describe cómo utilizar bucles condicionales para decidir si ejecutar o no código y cuándo hacerlo. Esta lección también describe cómo decidir el uso de una instrucción **Exit** para salir de un bucle.

Estructura de la lección

Esta lección incluye los siguientes temas y actividades:

- Cómo utilizar instrucciones **For...Next**
- Cómo utilizar instrucciones **For Each...Next**
- Cómo utilizar instrucciones **Do...Loop**
- Cómo utilizar instrucciones **Exit**
- Práctica: diseño de estructuras de bucle

Objetivos de la lección

En esta lección, aprenderá a:

- Utilizar instrucciones **For...Next** para ejecutar código un número determinado de veces.
- Utilizar instrucciones **For Each...Next** para ejecutar código para todos los elementos de una colección.
- Utilizar instrucciones **Do...Loop** para determinar la ejecución de un bloque de código dependiendo de una condición.
- Decidir si utilizar o no una instrucción **Exit** para salir inmediatamente de un bucle.

Cómo utilizar instrucciones For...Next

- Se utilizan cuando conocemos el número de veces que deseamos que se repita la ejecución de un código

```
For NamePos = 0 to 4
    MsgBox.Show(Names(NamePos))
Next
' In reverse order
For NamePos = 4 to 0 Step -1
    MsgBox.Show(Names(NamePos))
Next
```

Introducción

Podemos utilizar un bucle **For...Next** cuando conocemos el número de veces que es necesario que se ejecute un bucle. Una instrucción **For...Next** repite un conjunto de instrucciones un número específico de veces.

Sintaxis

Un bucle **For...Next** se ejecuta un determinado número de veces fijado por un contador de bucles. El valor del contador de un bucle **For...Next** puede incrementarse o disminuir dependiendo de si *step* es positivo o negativo.

La sintaxis de una instrucción **For...Next** es la siguiente:

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


Ejemplo

En el siguiente ejemplo, se utiliza una instrucción **For...Next** para recorrer el contenido de una pequeña matriz. Se utiliza un cuadro de mensaje para mostrar cada nombre en pantalla. Los nombres se muestran otra vez, pero en orden inverso.

```
Dim NamePos as Integer
Dim Names(4) As String
Names(0) = "Bob"
Names(1) = "Tina"
Names(2) = "Robert"
Names(3) = "George"
Names(4) = "Greg"

For NamePos = 0 to 4
    MessageBox.Show(Names(NamePos))
Next
' In reverse order
For NamePos = 4 to 0 Step -1
    MessageBox.Show(Names(NamePos))
Next
```

Anidar instrucciones For...Next

Podemos utilizar tantas instrucciones **For...Next** como sea necesario en una estructura **For...Next**. Añadir una o más instrucciones **For...Next** en otra instrucción **For...Next** se denomina *anidar*.

Pensemos en un escenario en el que deseemos crear múltiples puntos con coordenadas X e Y incrementales en un juego de ordenador. A continuación, utilizaremos esos puntos para asignar posiciones para gráficos.

Para crear estos puntos, podemos utilizar instrucciones **For...Next** anidadas. El código del ejemplo siguiente crea tres versiones de *myPoint* con valores X e Y incrementados, y nueve versiones de *otherPoint* con valores X e Y incrementados.

```
Private Sub CreatePoints( )
    Dim x, y As Integer
    Dim myPoint(7) As Point
    Dim otherPoint(7) As Point
    For x = 1 To 3
        myPoint(x).X += 1
        myPoint(x).Y += 1
        MessageBox.Show(myPoint(x).ToString)

        For y = 1 To 3
            otherPoint(y).X += y + 1
            otherPoint(y).Y += y + 2
            MessageBox.Show(otherPoint(y).ToString)
        Next
    Next
End Sub
```

Nota Los cuadros de mensaje de este código se proporcionan para que podamos observar los valores si decide probar este código. No serían necesarios en el anterior escenario de un juego de ordenador.

Cómo utilizar instrucciones For Each...Next

- Una colección es un conjunto de objetos agrupados juntos y a los que se hace referencia como una unidad. Por ejemplo:
 - Elementos de un cuadro de lista forman parte de una colección de **Elementos**
 - Un formulario tiene una colección de **Controles** que representan todos los controles de ese formulario
- Las instrucciones For Each ... Next se utilizan para recorrer los elementos de una colección

```
Sub LightBlueBackground (. . .)
    Dim ThisControl As System.Windows.Forms.Control
    For Each ThisControl In ThisForm.Controls
        ThisControl.BackColor = System.Drawing.Color.LightBlue
    Next ThisControl
End Sub
```

Introducción

Una instrucción **For Each...Next** ejecuta un bloque de instrucciones para cada elemento de una colección o una matriz.

Definición de colecciones

Una *colección* es un conjunto de objetos agrupados conjuntamente y a los que se hace referencia como una unidad.

Las colecciones nos permiten fácilmente añadir, borrar y contar elementos. Utilizando un número de índice o valores clave, podemos añadir elementos antes o después de otros elementos. Las colecciones son similares a las matrices, pero proporcionan funcionalidades que las matrices no tienen, incluyendo la facilidad de añadir, eliminar y manipular elementos, y más funcionalidades con objetos.

Podemos crear nuestras propias colecciones y añadirles nuestros propios elementos o elementos existentes. Un elemento puede ser cualquier tipo de datos, incluyendo objetos o estructuras, e incluso otros objetos de la colección.

El .NET Framework también proporciona numerosas colecciones predefinidas. Algunos ejemplos de colecciones del .NET Framework predefinidas son:

- Cuando añadimos elementos a un **ListBox**, utilizamos la colección **Items**. Cada elemento que añadimos es un miembro de la colección.
- Cada formulario tiene una colección de controles **Controls** que representa todos los controles de ese formulario. Podemos obtener una referencia a un control de la colección utilizando su índice, y recorrer los miembros de la colección utilizando las instrucciones **For Each...Next**.
- Cada hoja de cálculo de Microsoft Excel forma parte de la colección **Worksheet** de Excel. Esto significa que podemos escribir código para recorrer hojas de cálculo de Excel y reunir datos en una nueva hoja de cálculo.

Sintaxis

La sintaxis para la instrucción **For Each...Next** es la siguiente:

```
For Each elementvariable In collection
    ' Statement block to be executed for each value
    '   of elementvariable
Next [elementvariable]
```

Ejemplo

En el siguiente ejemplo, la instrucción **For Each...Next** se utiliza para recorrer un conjunto de controles de un formulario y establecer el color de fondo de cada control como azul claro (*light blue*):

```
Sub LightBlueBackground (ByVal ThisForm As _
    System.Windows.Forms.Form)
    Dim ThisControl As System.Windows.Forms.Control
    For Each ThisControl In ThisForm.Controls
        ThisControl.BackColor = System.Drawing.Color.LightBlue
    Next ThisControl
End Sub
```

Cómo utilizar instrucciones Do...Loop

■ Do...Loop Until

- Ejecuta el código del bucle y evalúa la condición. Repite hasta que la condición se evalúa como **True**.

■ Do Until...Loop

- Ejecuta el código en el bucle sólo si la condición se evalúa como **False**, y repite hasta que la expresión sea **True**.

■ Do...Loop While

- Ejecuta el código en el bucle y evalúa la condición. Repite hasta que la condición sea **False**.

■ Do While...Loop

- Ejecuta el código en el bucle sólo si la condición se evalúa como **True**, y repite hasta que la expresión sea **False**.

Introducción

Las instrucciones **Do...Loop** proporcionan un modo para que nuestra aplicación ejecute un bloque de código mientras una condición sea verdadera o hasta que lo sea. Podemos utilizar instrucciones **Do...Loop** cuando conocemos la condición que determina cuando debe detenerse la ejecución de nuestro código.

Tipos de instrucciones Do...Loop con la palabra clave Until

Cuando añadimos la palabra clave **Until** a una instrucción **Do...Loop**, ordenamos a nuestro programa que haga algo hasta que una condición sea verdadera. En las instrucciones **Do...Loop**, una expresión puede ser evaluada al final o al principio del bucle. Las estructuras de bucle que evalúan expresiones al final de un bucle aseguran que el código del bucle se ejecuta al menos una vez. Las estructuras de bucle que evalúan expresiones al principio de un bucle pueden no ser ejecutadas, dependiendo del valor inicial de la expresión.

Si desea verificar la condición que está probando antes de ejecutar el código, y desea que el código se ejecute hasta que una condición sea verdadera, puede utilizar la instrucción **Do Until...Loop**. Esta instrucción ejecuta el código en el ciclo únicamente si la condición es **False** y se repite hasta que la expresión evaluada sea **True**.

Si desea verificar la condición que está probando después de ejecutar el código, y desea que el código se ejecute hasta que una condición sea verdadera, puede utilizar la instrucción **Do...Loop Until**. Esta instrucción ejecuta y repite la ejecución hasta que la expresión sea **True**.

Sintaxis

La sintaxis **Do Until...Loop** es como sigue:

```
Do [Until condition]
  [statements]
[Exit Do]
Loop
```

La sintaxis **Do...Loop Until** es como sigue:

```
Do
    [statements]
[Exit Do]
    [statements]
Loop [While condition]
```

Ejemplos

En el siguiente ejemplo de uso de **Do...Loop Until**, se recorrerá el código hasta que el usuario haga clic en **No** en un cuadro de mensaje:

```
Dim LoopCount As Integer
Do
    LoopCount = LoopCount + 1
Loop Until MessageBox.Show("Loop?", "Do... Loop Until", _
    MessageBoxButtons.YesNo) = DialogResult.No
```

El siguiente ejemplo de utilización de **Do Until...Loop** encuentra todos los archivos de un directorio que concuerdan con una especificación de archivo. El ejemplo utiliza la función **Dir** para devolver el primer nombre de archivo que concuerde con la ruta especificada. La función **Dir** se invoca de nuevo para devolver los nombres de archivo adicionales que concuerdan con la ruta. Cuando no haya más nombres de archivo que concuerden, la condición **Do Until...Loop** se evaluará como **True**, y la longitud de la cadena devuelta por **Dir** será igual a cero:

```
Dim Message As String
Dim Filespec As String
Dim Match As String

Message = "Enter a file specification."
' Get file extension
Filespec = InputBox(Message)
' Find first match
Match = Dir(Filespec)
Do Until Len(Match) = 0
    ' Display matching files
    MessageBox.Show(Match)
    ' Find next match
    Match = Dir( )
Loop
MessageBox.Show ("There are no more matching files.")
```

Uso de While en lugar de Until

Podemos utilizar la palabra clave **While** en lugar de la palabra clave **Until** cuando la condición para que el bucle continúe sea **True** en lugar de **False**. Por ejemplo, los dos bloques de código siguientes son funcionalmente equivalentes:

```
Dim EndOfLoop As Boolean
EndOfLoop = False
Do
' . . .
Loop Until EndOfLoop

Dim EndOfLoop As Boolean
EndOfLoop = False
Do
' . . .
Loop While Not(EndOfLoop)
```

Cómo utilizar instrucciones Exit

- Se utilizan para salir inmediatamente de bucles Do o bucles For cuando se cumple una condición

```
Do Until y = -1
  If x < 0 Then Exit Do
  x = Sqrt (x)
  If y > 0 Then Exit Do
  y = y + 3
  If z = 0 Then Exit Do
  z = x / y
Loop
```

Introducción

Es posible que en una estructura de bucle, deseemos salir del bucle inmediatamente si se cumple una determinada condición. Podemos utilizar las instrucciones **Exit** para salir de un bucle **Do** o de un bucle **For**. Las instrucciones **Exit** permiten salir directamente de cualquier estructura de decisión, bucle o procedimiento. Transfieren de forma inmediata la ejecución a la instrucción que sigue a la última instrucción de control.

Sintaxis

La sintaxis para el uso de la instrucción **Exit** en una instrucción **Do...Loop** es la siguiente:

```
Exit Do
```

La sintaxis para el uso de la instrucción **Exit** en una instrucción **For...Next** es la siguiente:


```
Exit For
```

Ejemplo

El siguiente ejemplo muestra varias incidencias de la instrucción **Exit** dentro de una instrucción **Do...Loop**:

```
Do Until y = -1
  If x < 0 Then Exit Do
  x = Sqrt (x)
  If y > 0 Then Exit Do
  y = y + 3
  If z = 0 Then Exit Do
  z = x / y
Loop
```

Práctica: Diseño de estructuras de bucle



- En cuatro escenarios, decidir qué estructura de bucle se utilizaría para solucionar un problema
- En el entorno de desarrollo, utilice instrucciones For...Next para crear una tabla de multiplicación

En esta práctica, decidiremos qué estructuras de bucle son adecuadas para cuatro escenarios distintos de programación. A continuación, utilizará instrucciones **For...Next** para crear una tabla de multiplicación.

Escenario 1

Hemos creado una aplicación de caja registradora que almacena en una matriz el precio de cada artículo adquirido. Cuando el usuario finaliza su compra, el código debe buscar en la matriz y calcular el total de todos los artículos comprados. El tamaño de la matriz variará según el número de artículos adquiridos.

¿Qué estructura de bucle será la más adecuada para solucionar este problema?

For Each...Next.

Una instrucción For Each...Next recorrería todos los elementos de la matriz, proporcionando cada elemento de la matriz en la variable For Each.

Escenario 2

Nuestra aplicación contiene la ecuación matemática $y = x^3 + 3$. La aplicación necesita calcular el resultado de esta ecuación para valores de x en el intervalo -100 a 100 en pasos de 10.

¿Qué estructura de bucle será la más adecuada para solucionar este problema?

For...Next.

For x = -100 To 100 Step 10

Next

Este problema también podría solucionarse con una instrucción Loop Until...Do y una variable incrementada, pero una instrucción For...Next es más fácil de leer.

Escenario 3

Nuestra aplicación usa el contenido de un archivo de registro del disco duro de un equipo y muestra estos datos en el cuadro de texto de un formulario. El programa lee secuencialmente las líneas de datos guardadas en el archivo de registro. El número de líneas de datos almacenadas en el archivo es desconocido, pero tenemos una función denominada **EOF** que devuelve **True** cuando se llega al final del archivo.

¿Qué estructura de bucle será la más adecuada para solucionar este problema?

Do Until...Loop.

Como no se conoce el número de líneas que contiene el archivo, una instrucción Do Until...Loop es ideal, especialmente porque en este caso hay disponible una expresión para determinar si se ha llegado al final del archivo.

FileOpen(1, "C:\file.log", OpenMode.Input)

Do Until EOF(1)

**TextBox1.Text=TextBox1.Text & Chr(13) & Chr(10) & _
 LineInput(1)**

Loop

FileClose(1)

Escenario 4

Nuestra aplicación utiliza un objeto **City** para determinar los valores de descuento en las ventas de varios artículos, dependiendo de la ciudad desde la que se adquieren los artículos. Queremos mostrar los descuentos asociados a cada ciudad en un cuadro de lista.

¿Qué estructura de bucle será la más adecuada para solucionar este problema?

For Each...Next.

For Each City in CityCollection

ListBox.Items.Add(City.Discount)

Next

Uso de una instrucción
For...Next

✍ **Uso de una instrucción For...Next**

En esta sección de la práctica, abriremos un nuevo proyecto y utilizaremos una instrucción **For...Next** para crear una tabla de multiplicación en un formulario.

1. Abrir un nuevo proyecto en Visual Basic .NET. Utilizar la plantilla Aplicación para Windows. Nombrar el proyecto **MultiplicationTable** y seleccionar la carpeta donde quiera crearlo.
2. Añadir una etiqueta al formulario y modificar su tamaño para que el ancho sea **255** y la altura sea **190**. Centrar la etiqueta en el formulario.
3. Establecer la propiedad **Font** de la etiqueta como **Courier New**.
4. Crear un controlador de eventos para el evento **Form1_Load** y añadir el siguiente código:

```
Dim outerLoop, innerLoop As Integer
Dim result As String

' Outer For...Next loop
For outerLoop = 1 To 10
    ' Inner For...Next loop
    For innerLoop = 1 To 10
        result = result & " " & (outerLoop * innerLoop)
    Next innerLoop
    result = result & vbCrLf
Next outerLoop

' Display result in the label
Label1.Text = result
```

5. Responder a las siguientes cuestiones antes de ejecutar la aplicación:

a. ¿Qué tres líneas de código definen el bucle interior?

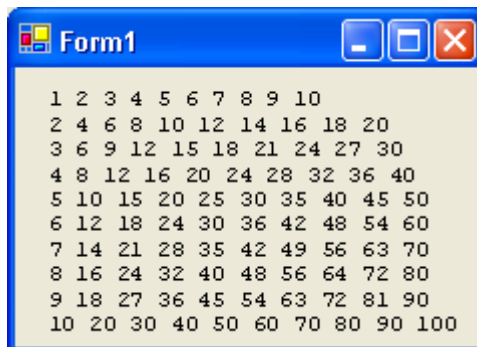
```
For innerLoop = 1 To 10
    result = result & " " & (outerLoop * innerLoop)
Next innerLoop
```

b. ¿Qué valores tendrán *outerLoop* e *innerLoop* la segunda vez que el bucle interior (*inner loop*) se ejecute?

El valor de *outerLoop* será 1; el valor de *innerLoop* será 2.

c. ¿Cómo espera que sea el resultado de la etiqueta después de que se ejecute el código?

La siguiente ilustración muestra el contenido de la etiqueta en la aplicación que se está ejecutando.



6. Ejecutar la aplicación para verificar su respuesta a la pregunta 5c.

Aparecerá una tabla de multiplicación en la etiqueta del formulario.

Archivos de solución

Los archivos de solución de esta práctica se encuentran en la carpeta MultiplicationTable\Solution dentro del archivo practs05.zip.