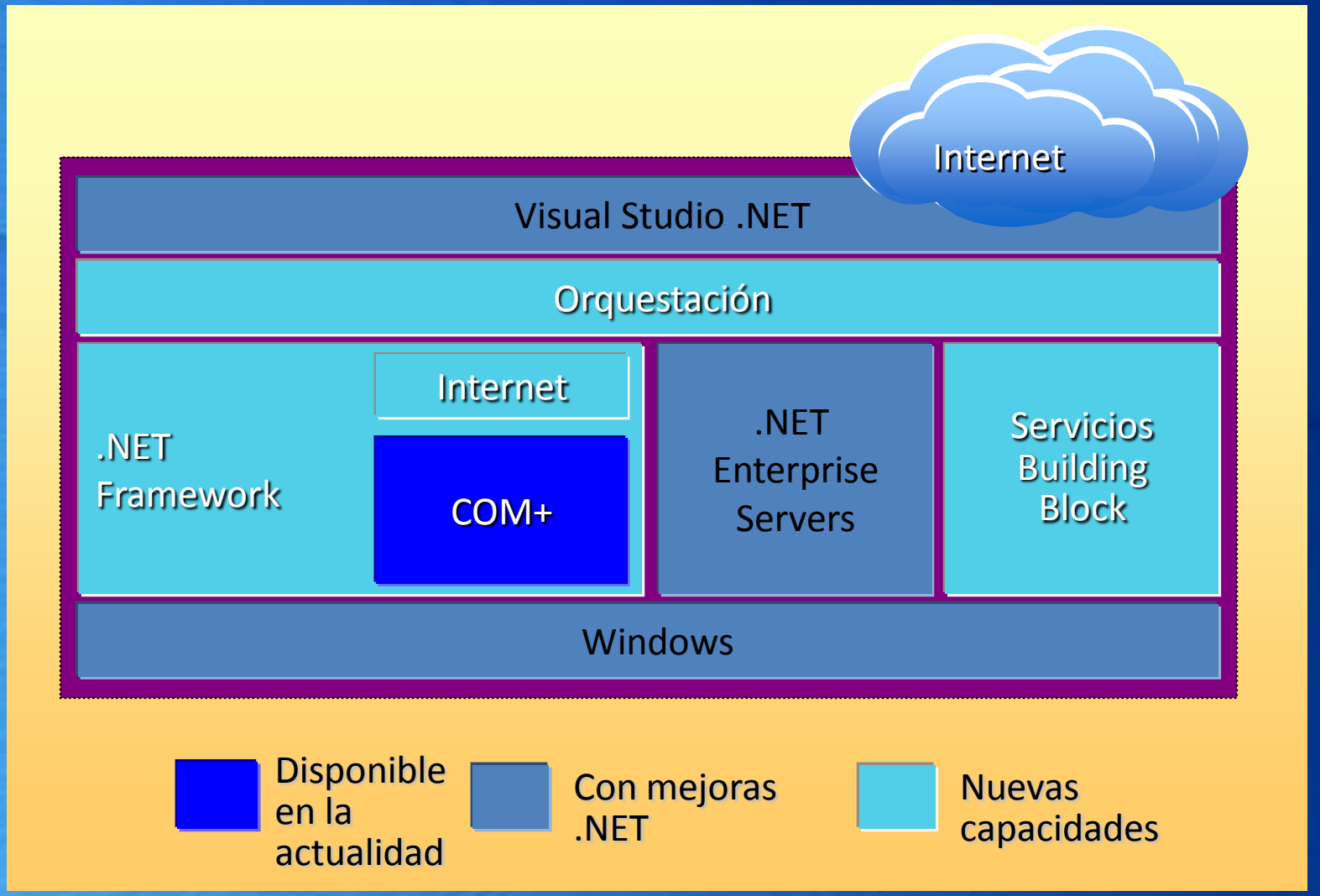
Curso de Visual Basic.Net

Conceptos básicos de .NET

¿Qué es la plataforma Microsoft .NET?

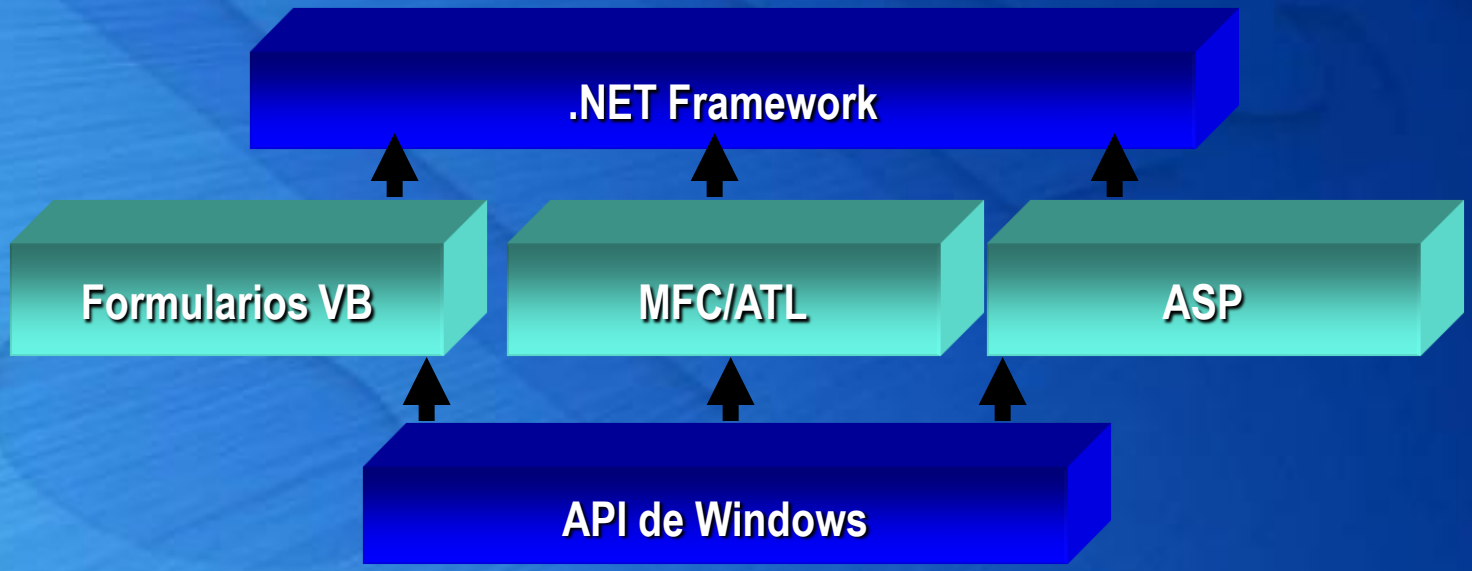


".NET Framework es un entorno para construir, instalar y ejecutar servicios Web y otras aplicaciones.
Se compone de tres partes principales: el Common Language Runtime, las clases Framework y ASP.NET"

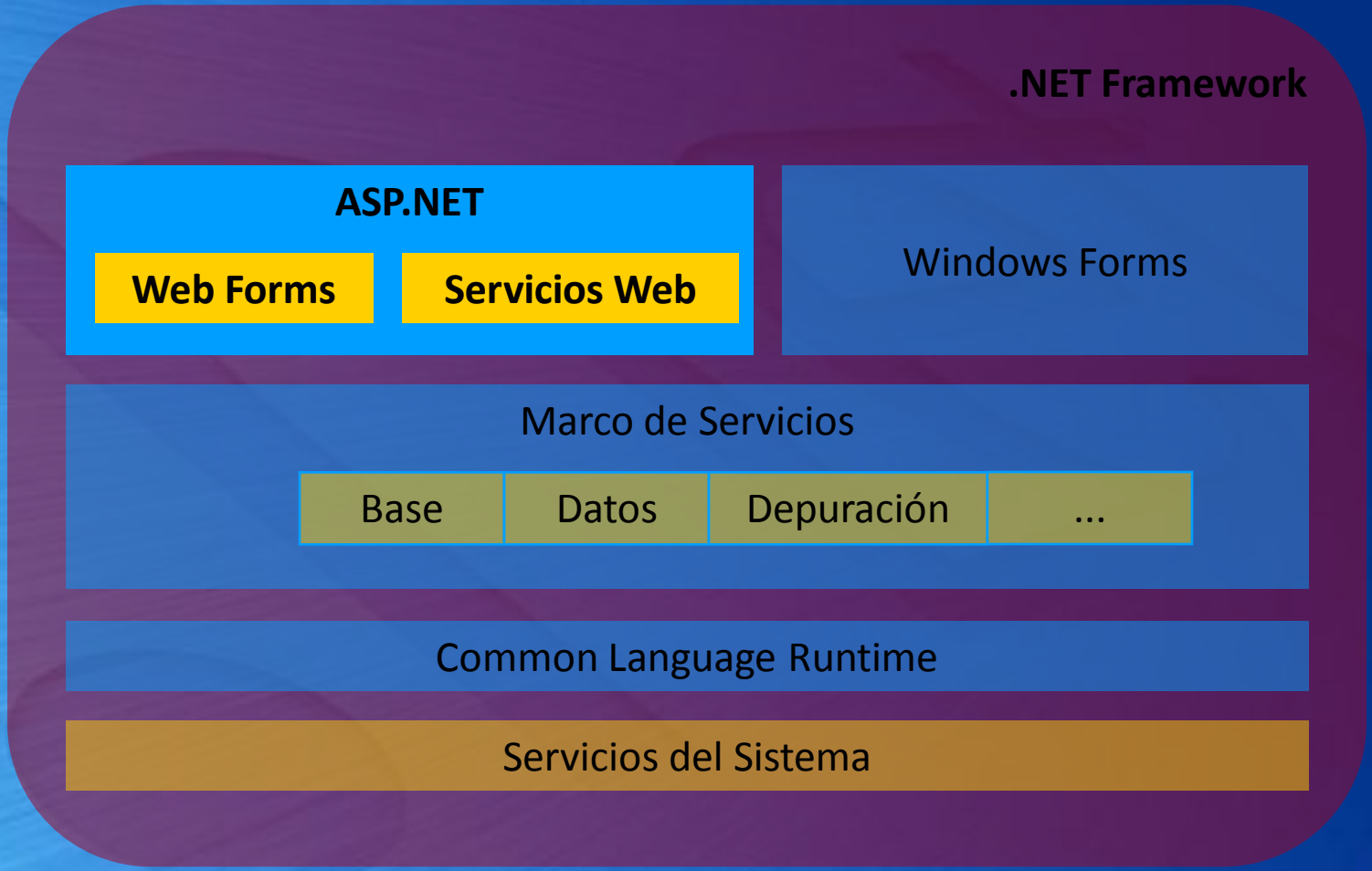
El .NET Framework es el corazón de .NET, cualquier cosa que queramos hacer en cualquier lenguaje .NET debe pasar por el filtro cualquiera de las partes integrantes del .NET Framework.

Beneficios del .NET Framework

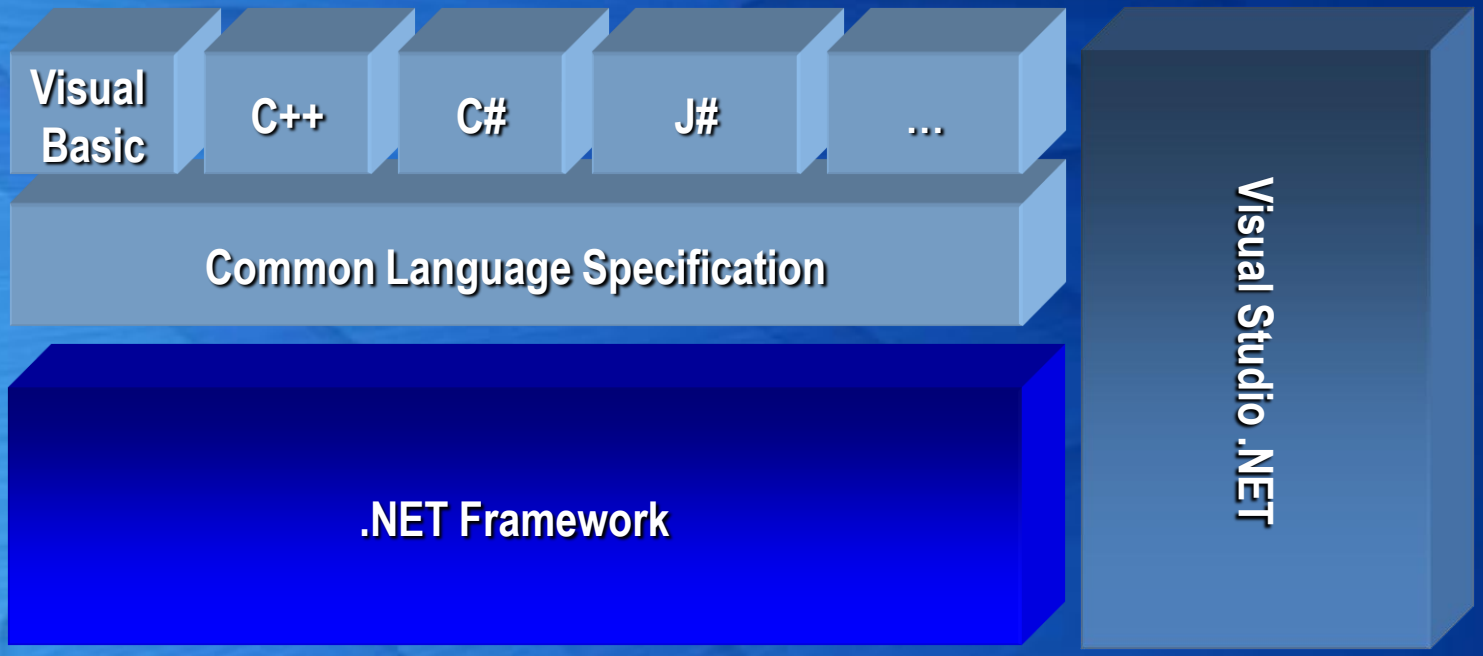
- Basado en estándares y prácticas Web
- Extensible
- Fácil de utilizar por los desarrolladores
- Diseñado utilizando modelos de aplicaciones unificados



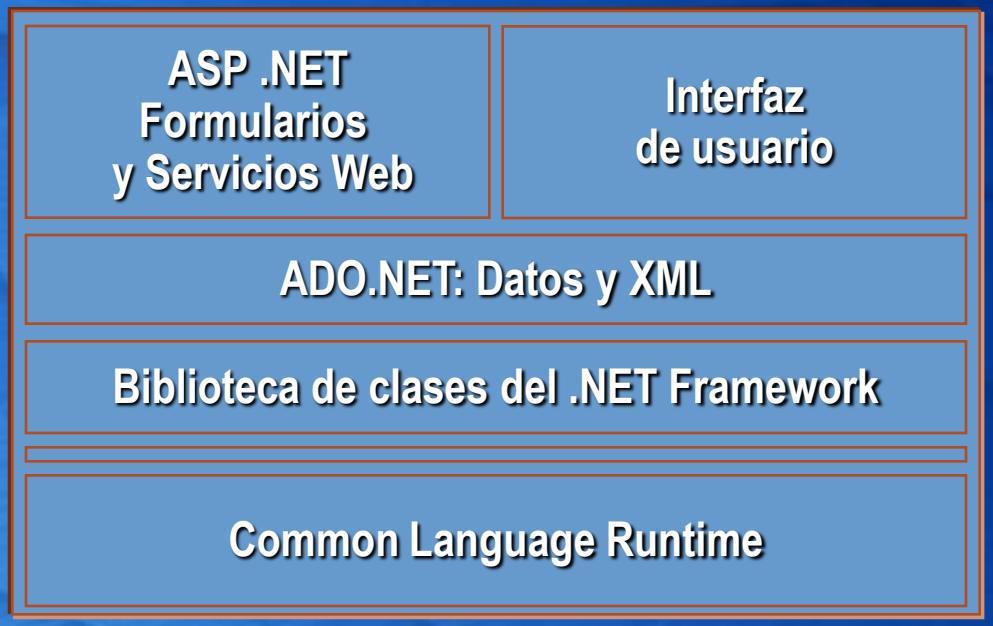
El .NET Framework- Arquitectura



Lenguajes de Net



Componentes de Net Framework

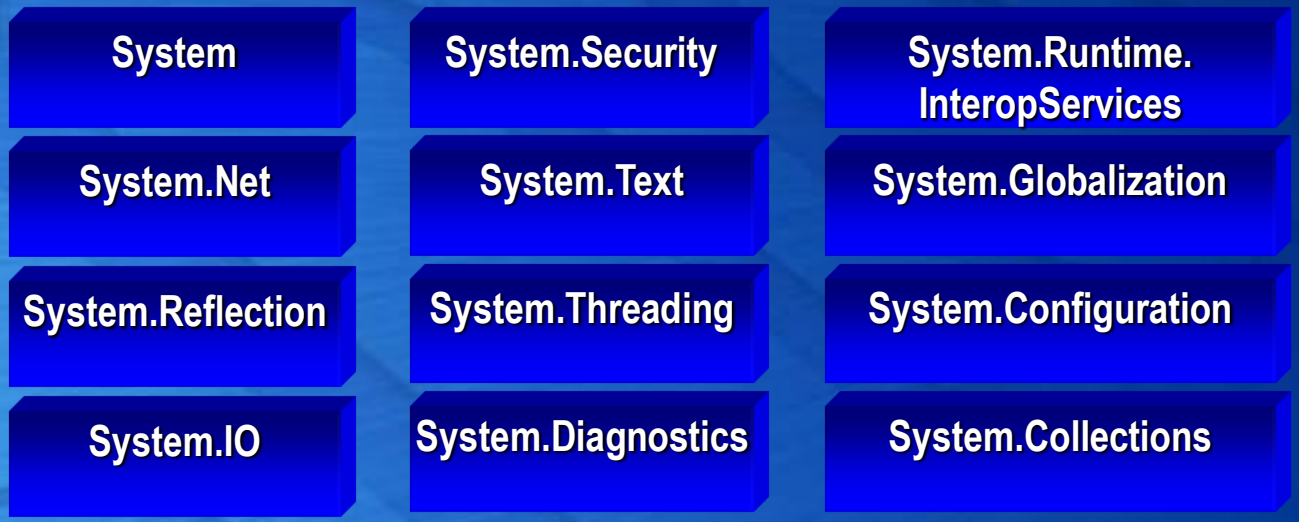


CLR

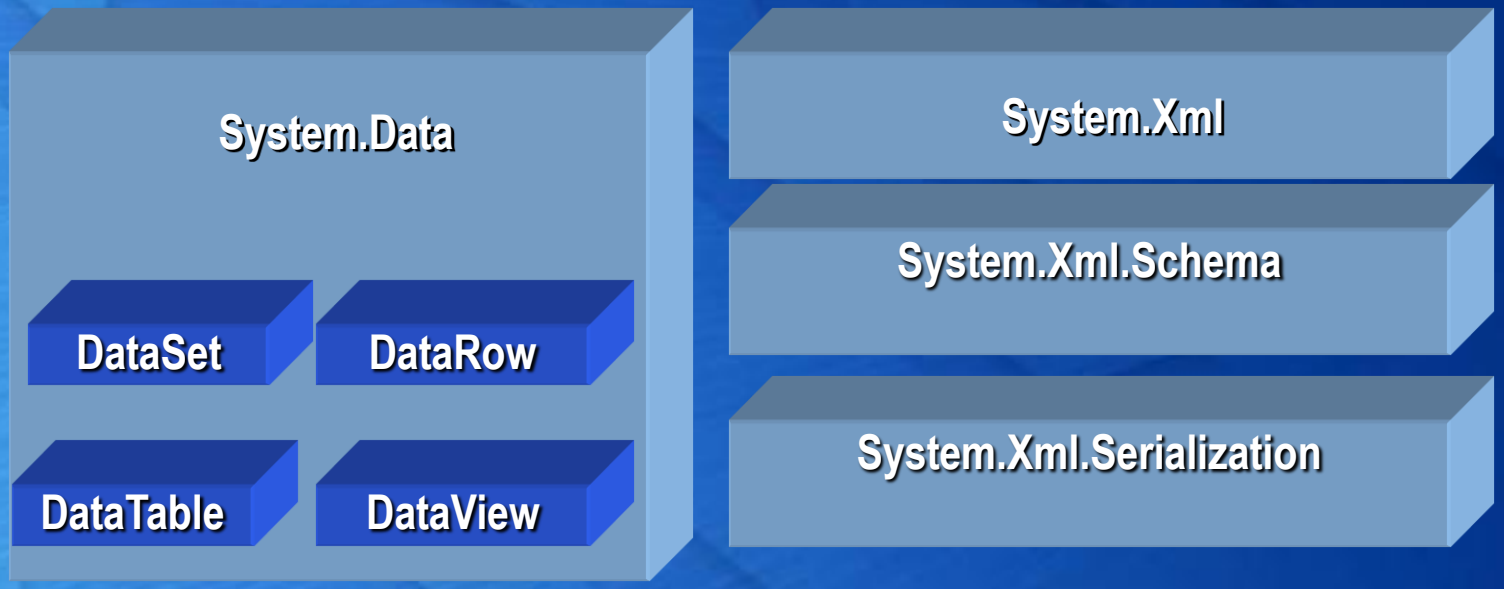
- El Common Lenguage Runtime (CLR) es una serie de librerías dinámicas (DLLs), también llamadas assemblies, que hacen las veces de las DLLs del API de Windows.
- Así como las librerías runtime de Visual Basic o C++.
- Cualquier ejecutable depende de una forma u otra de una serie de librerías, ya sea en tiempo de ejecución como a la hora de la compilación.
- Por otro lado, la librería de clases de .NET Framework proporcionan una jerarquía de clases orientadas a objeto disponibles para cualquiera de los lenguajes basados en .NET, incluido el Visual Basic.

Common Language Runtime

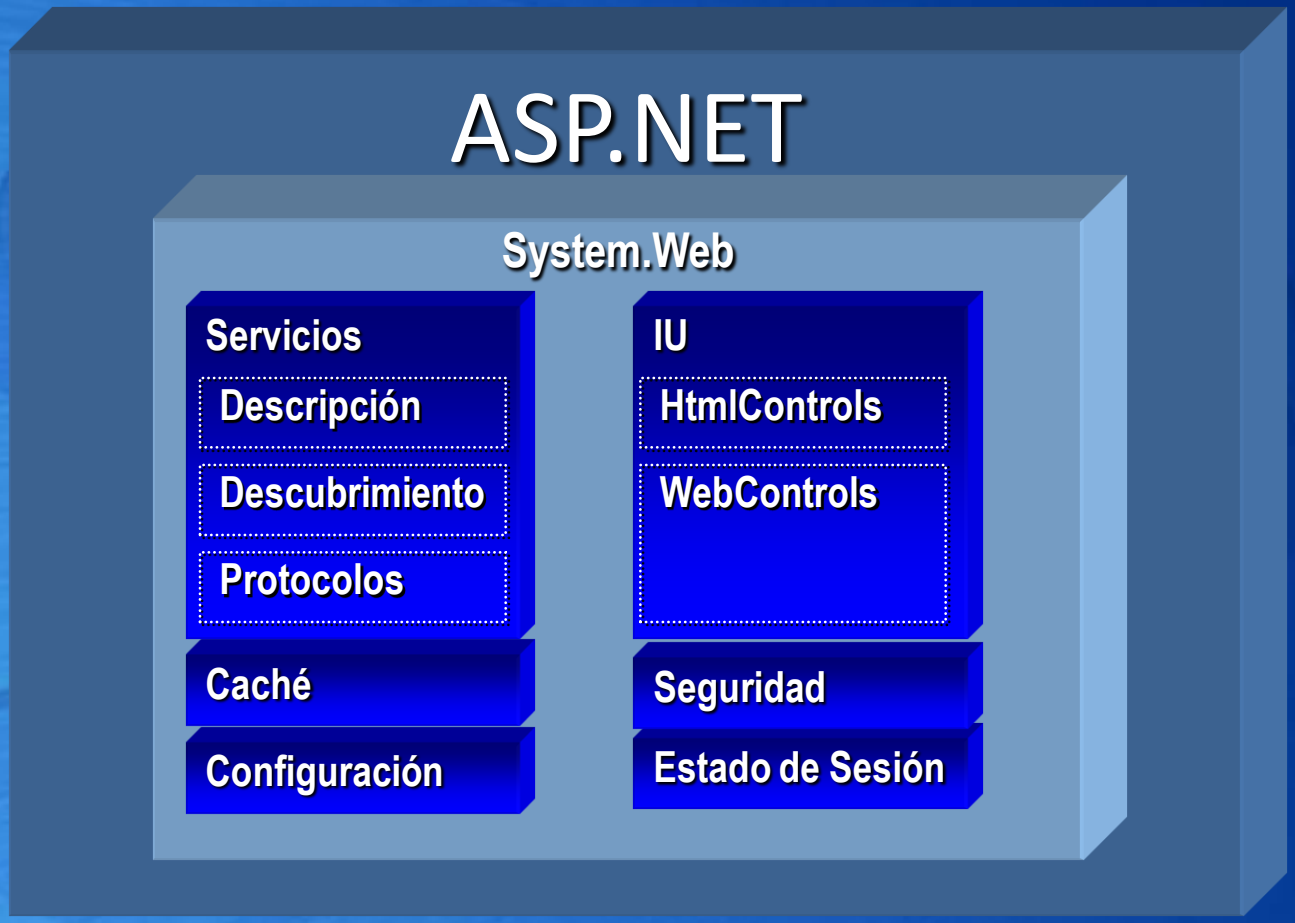
Biblioteca de Clases



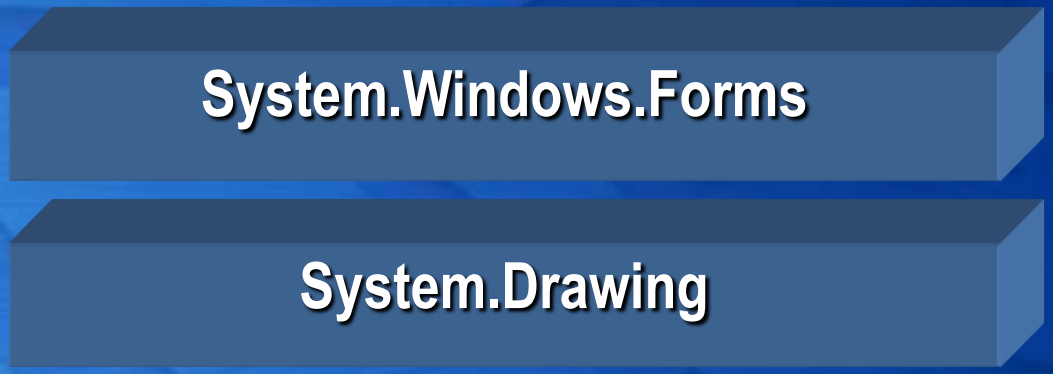
ADO .Net y XML


System.Data
DataSet
DataRow
DataTable
DataView
System.Xml
System.Xml.Schema
System.Xml.Serialization

ASP.NET: Formularios Web y Servicios Web



Interfaz de Usuario



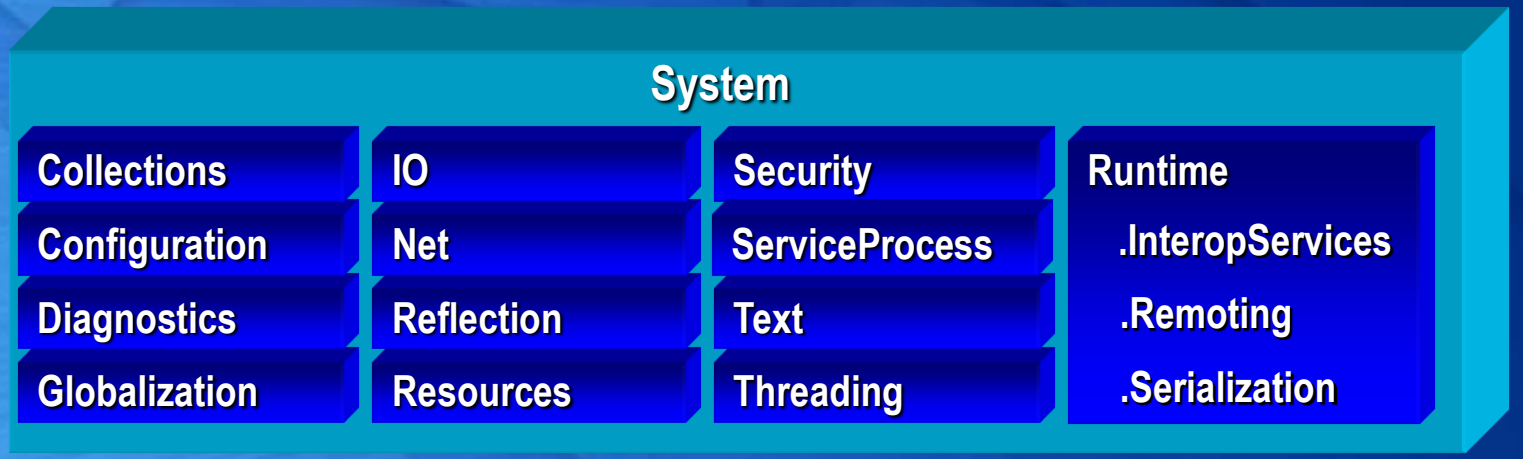
Mejoras en Visual Basic Net

- Principales mejoras del lenguaje
  - Soporte mejorado orientado a objetos
  - Gestión de excepciones estructurada
- Acceso total al .NET Framework
  - Nuevas opciones de manejo de hilos
  - Recolector de basura
- Desarrollo Web mejorado
  - Creación de Formularios Web tan fácilmente como formularios Windows
  - Crear servicios Web rápidamente

Descripción de los espacios de nombres

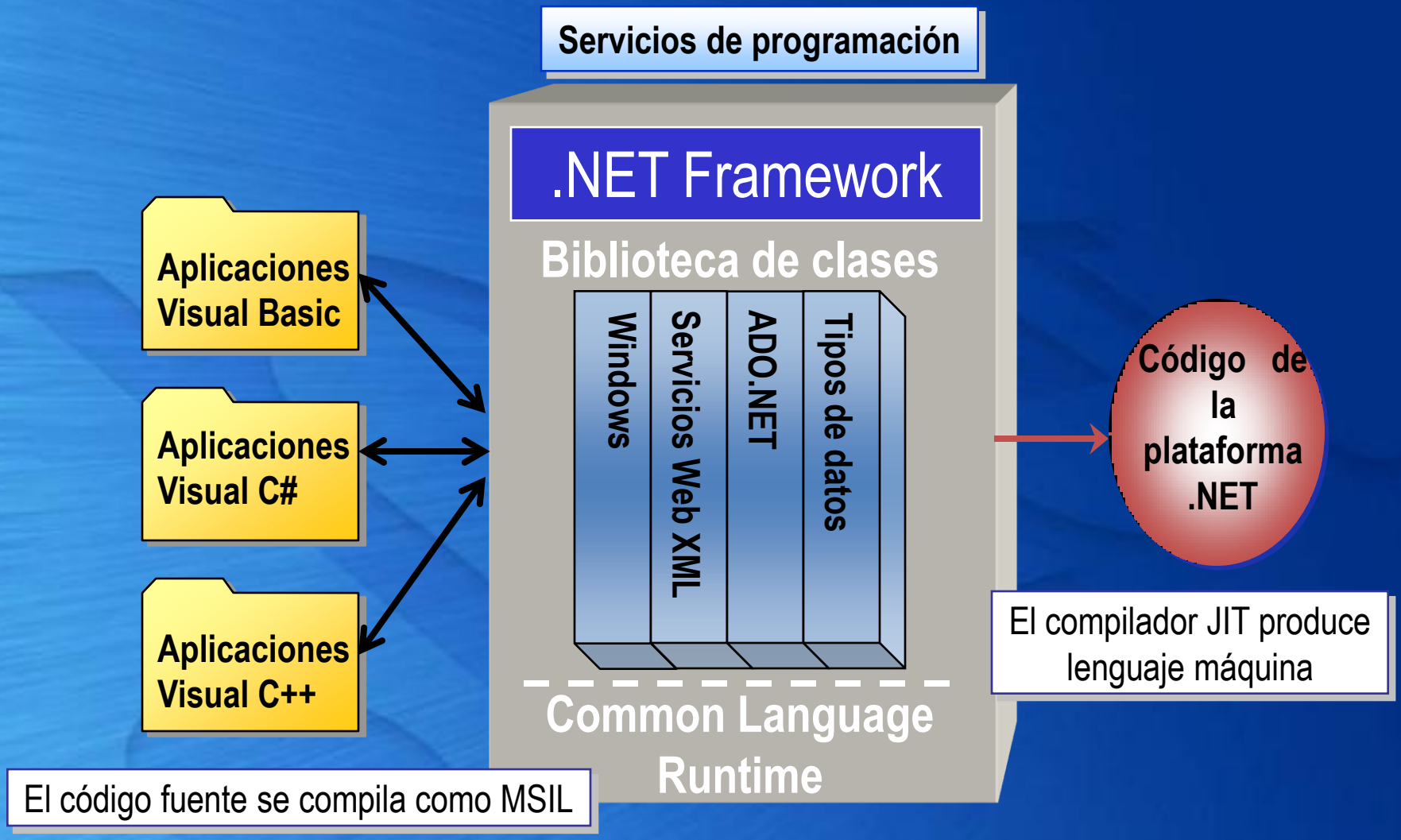
- Espacios de nombres
- Espacios de nombres utilizados en módulos obligatorios
- Espacios de nombres utilizados en módulos opcionales

Espacios de nombres



¿Qué es la plataforma .NET?

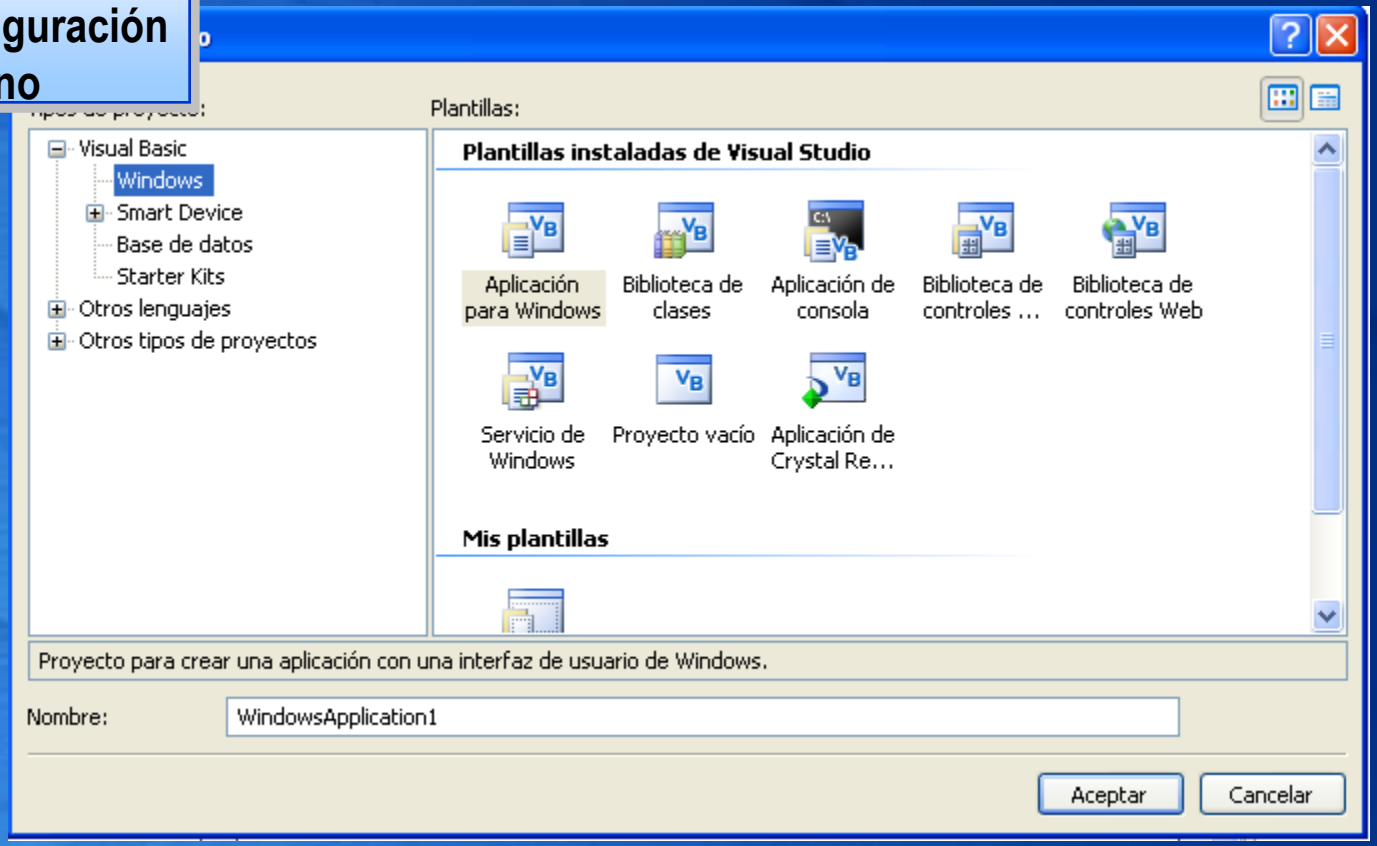
Cómo funciona el .NET Framework



¿Qué es Visual Studio .NET?

Qué es una plantilla de aplicación?

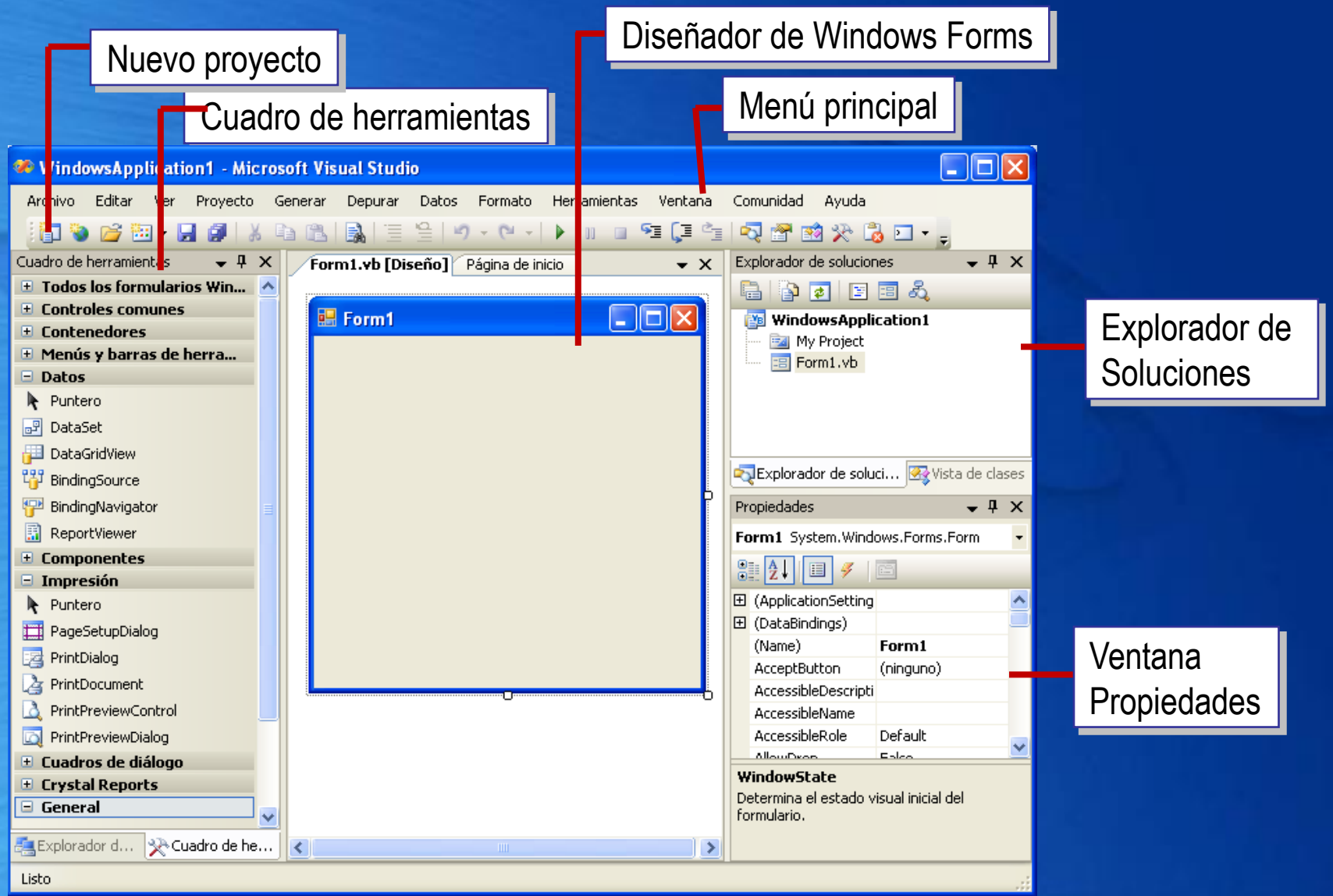
Proporciona archivos de inicio, estructura de proyecto y configuración del entorno



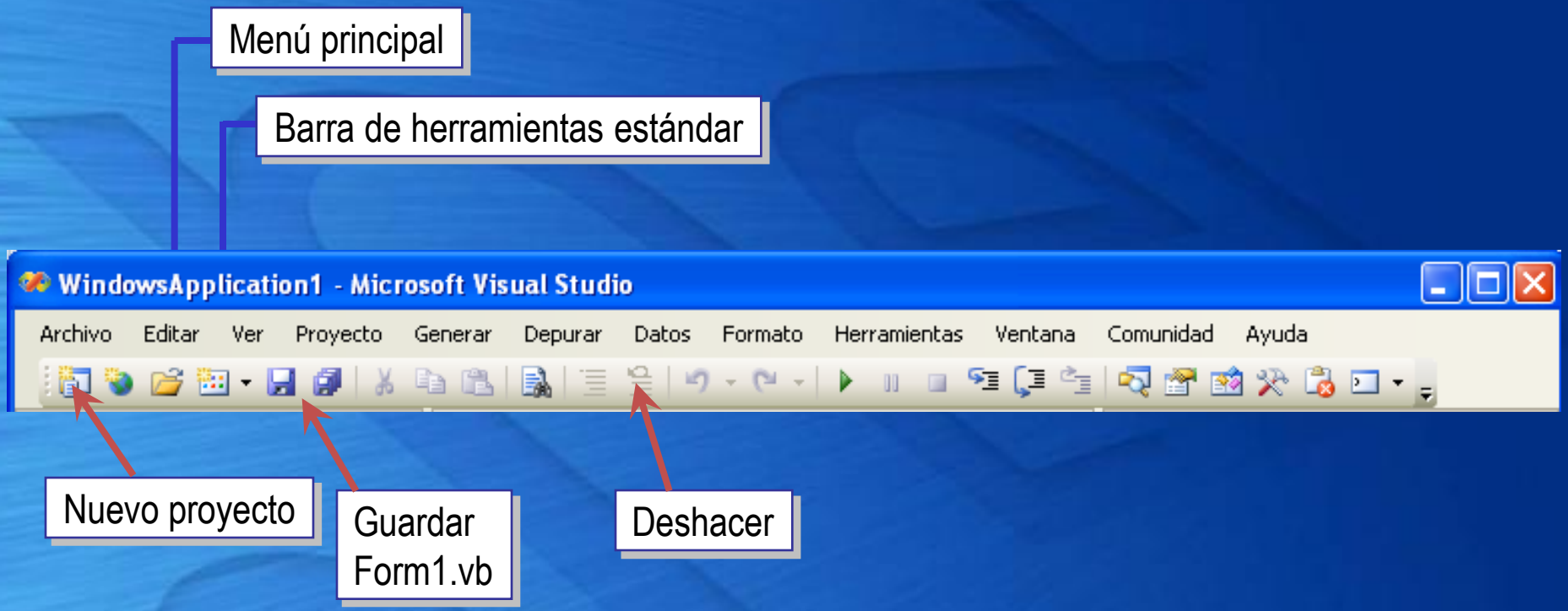
Trabajar con un proyecto Visual Basic .NET

1. Iniciar un nuevo proyecto Visual Basic .NET basado en la plantilla Aplicación para Windows
2. Ejecutar el proyecto dentro del entorno de desarrollo
3. Generar un archivo ejecutable
4. Ejecutar el proyecto fuera del entorno de desarrollo
5. Visualizar los archivos del proyecto en el Explorador de soluciones
6. Guardar y cerrar el proyecto

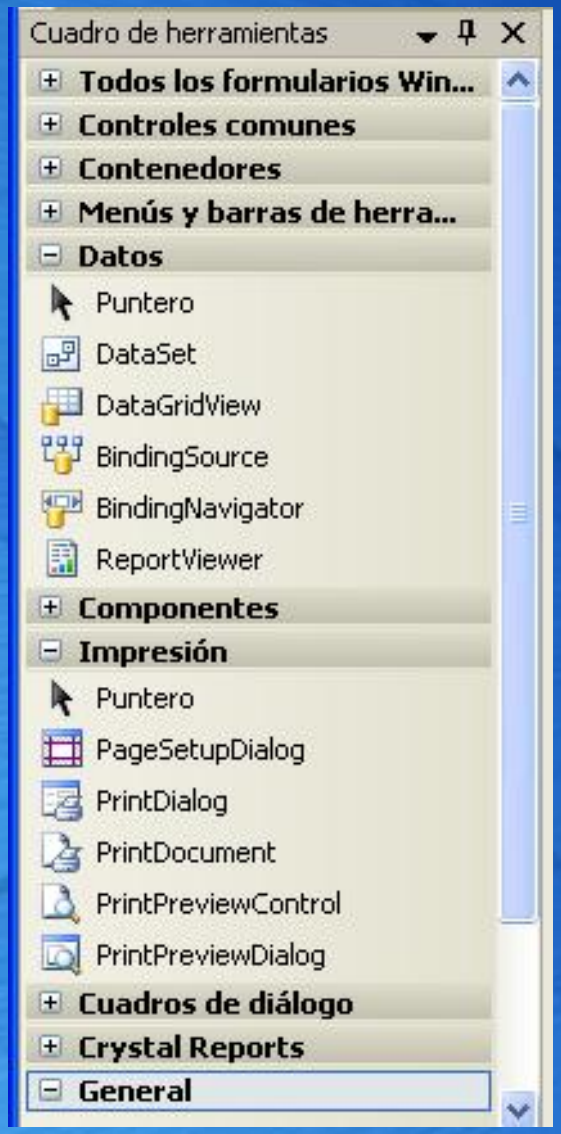
Entorno de desarrollo



Menús y barras de herramientas

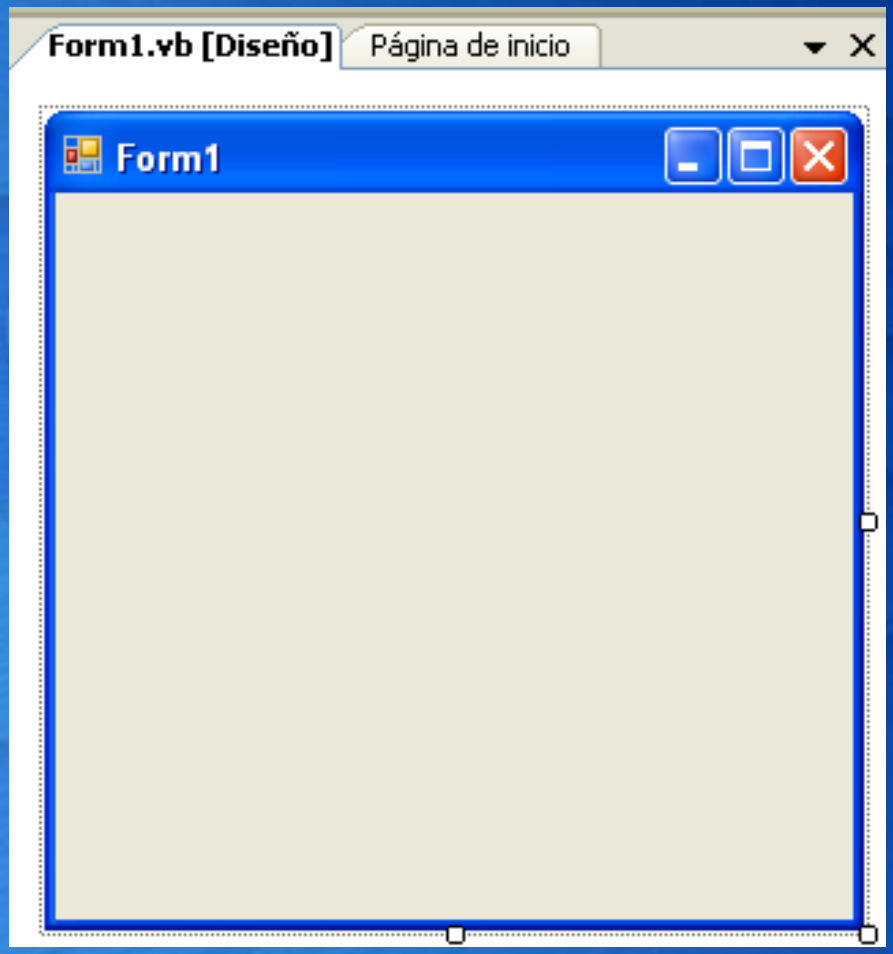


Cuadro de herramientas



Controles para crear el interfaz de usuario

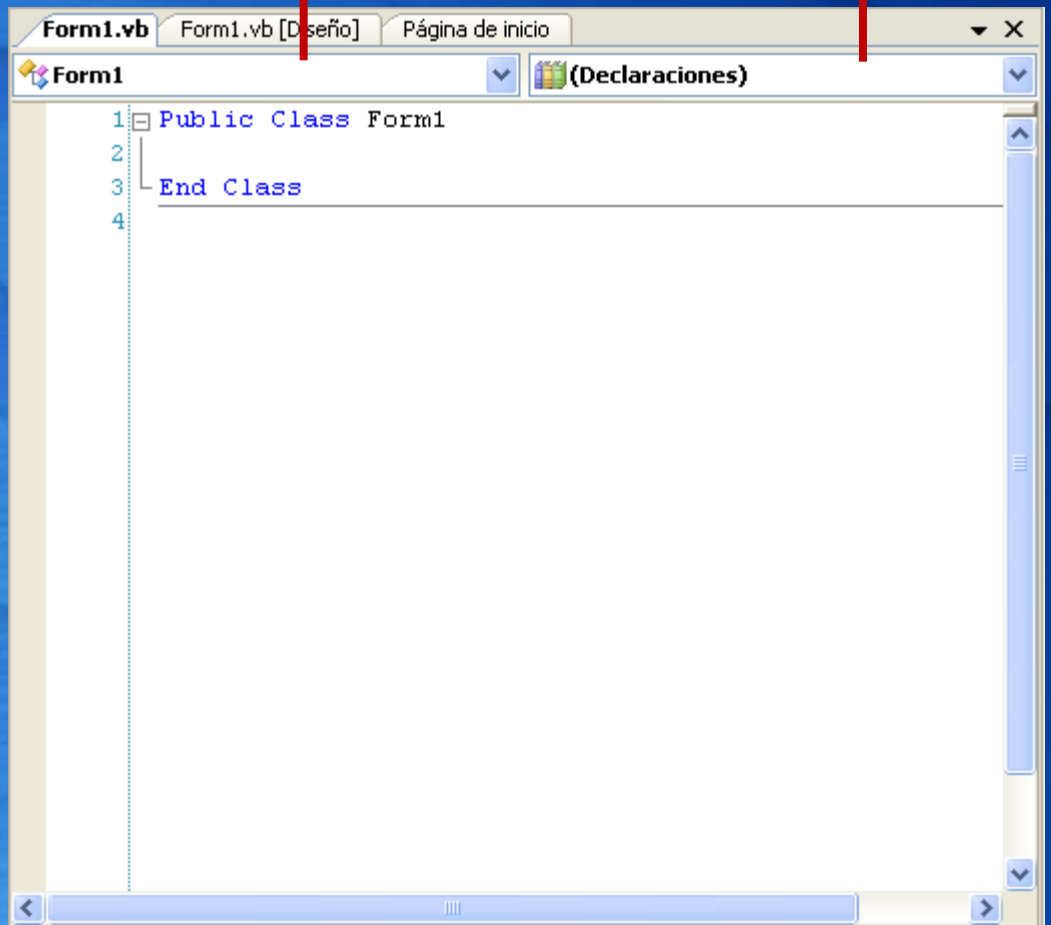
Diseñador de Windows Forms



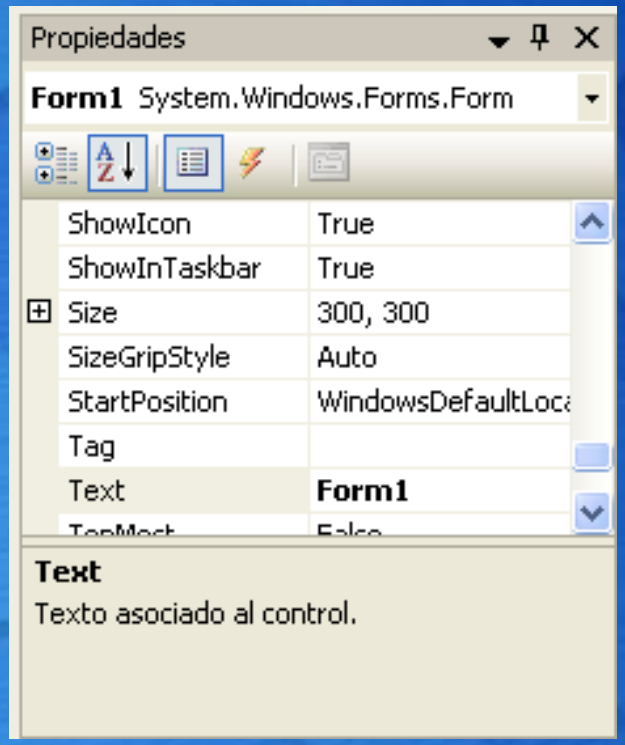
Editor de código

Lista de nombres de clases

Lista de nombres de métodos



Ventana Propiedades

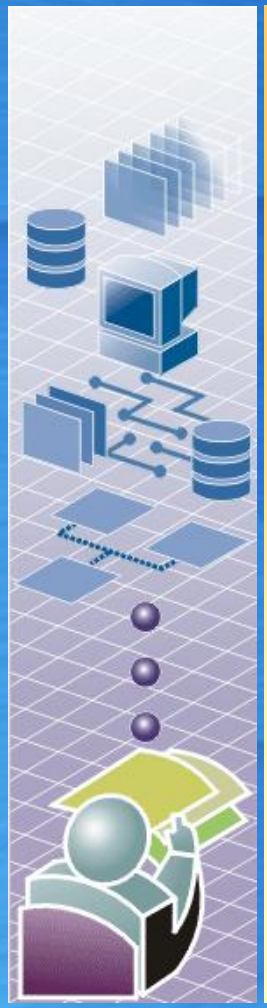


Establecer propiedades como tamaño, título y color

Otras ventanas de programación

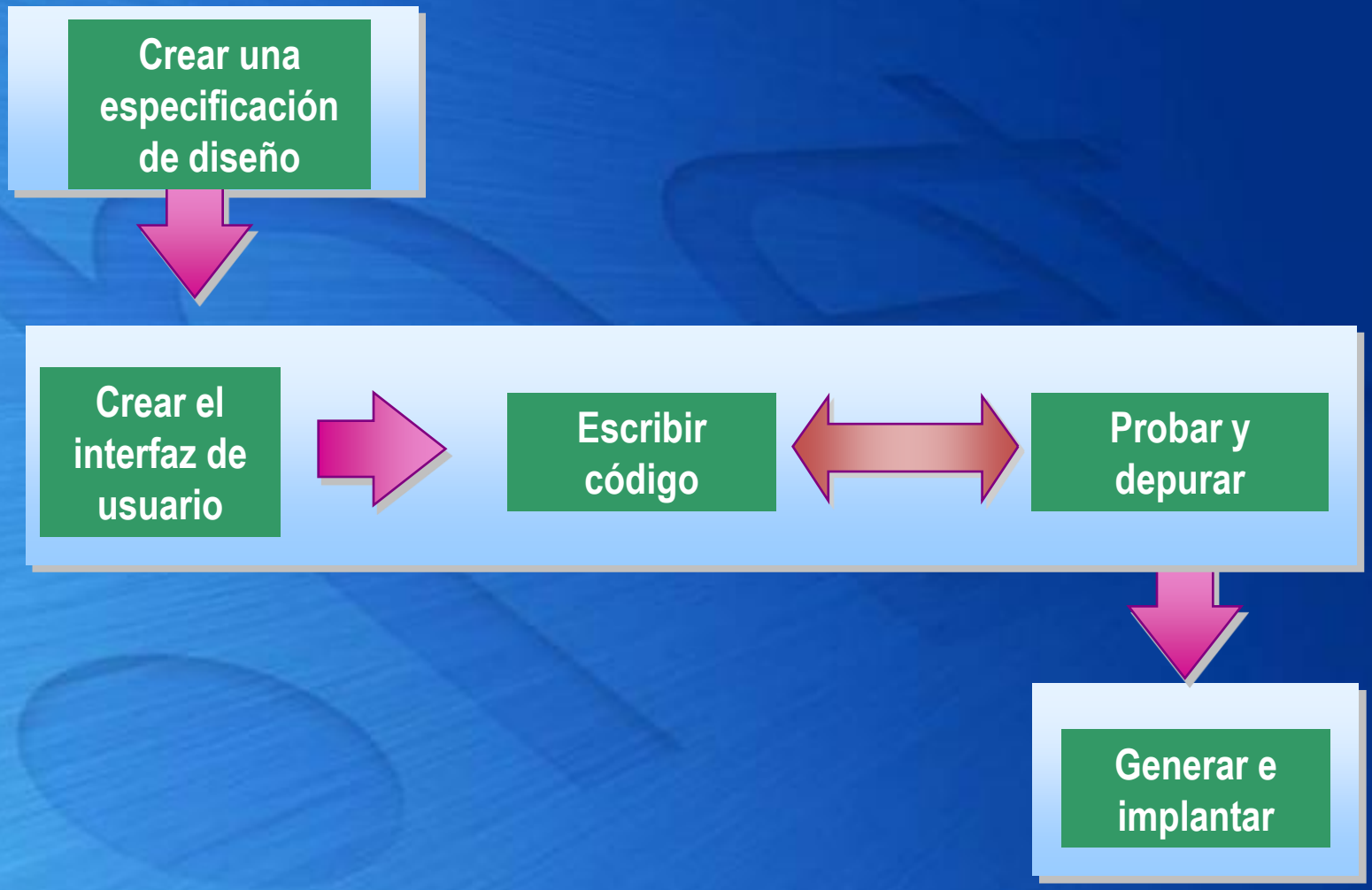
Ventana de programación	Objetivo
Lista de tareas	Ayuda a organizar y gestionar el trabajo de generar la aplicación
Resultados	Muestra mensajes de estado para varias características en el entorno de desarrollo
Vista de clases	Permite examinar el código tras las clases y navegar por los símbolos de la solución
Comandos	Permite emitir comandos o evaluar expresiones en el entorno de desarrollo
Examinador de objetos	Permite visualizar objetos y sus miembros

Práctica: Trabajar en el entorno de desarrollo



1. Abrir y ejecutar una aplicación existente
2. Examinar un formulario en el Diseñador de Windows Forms y el Editor de código
3. Abrir, cerrar, reabrir y ocultar el Cuadro de herramientas
4. Examinar la configuración de las propiedades de los controles
5. Utilizar la ventana Ayuda dinámica

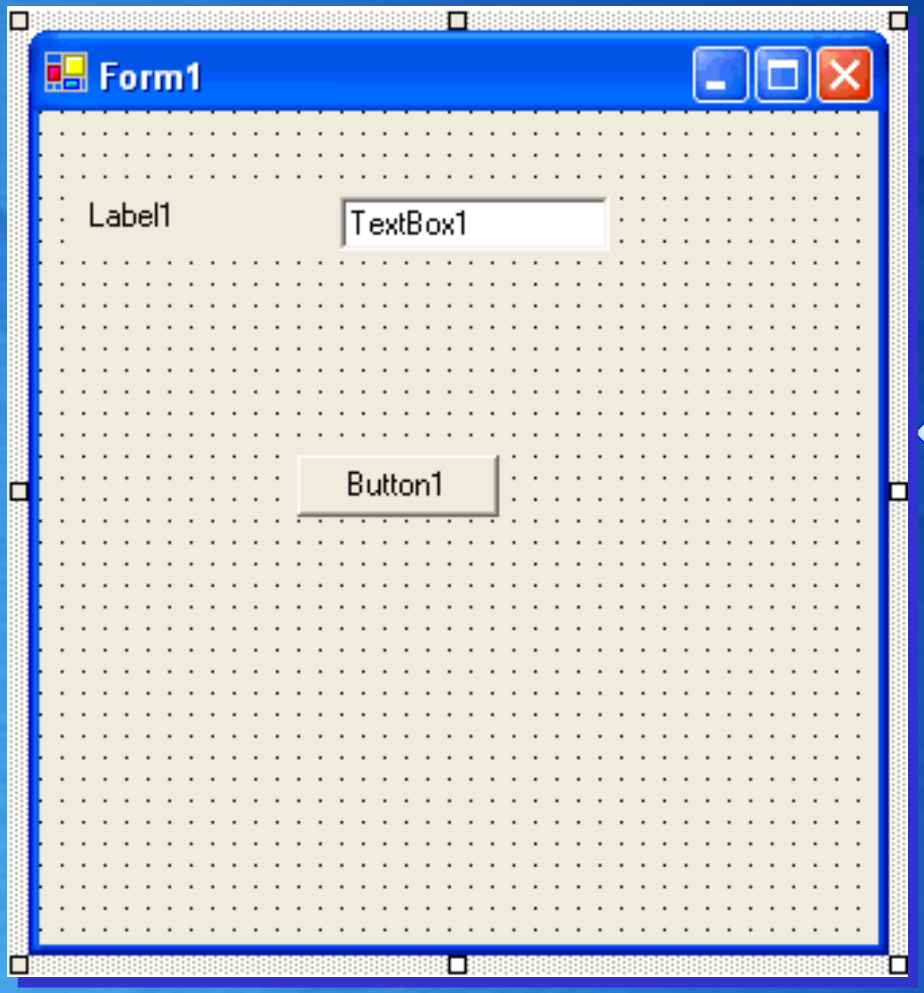
Crear un proyecto Visual Basic .NET



El proceso de desarrollo

1. Crear una especificación de diseño
2. Crear el interfaz de usuario
3. Establecer las propiedades de los objetos del interfaz de usuario
4. Escribir código para añadir funcionalidad
5. Probar y depurar la aplicación
6. Generar un archivo ejecutable
7. Crear una aplicación de instalación

Cómo crear el interfaz de usuario



Ubicar controles en el formulario desde el Cuadro de herramientas

Cómo establecer las propiedades de los controles

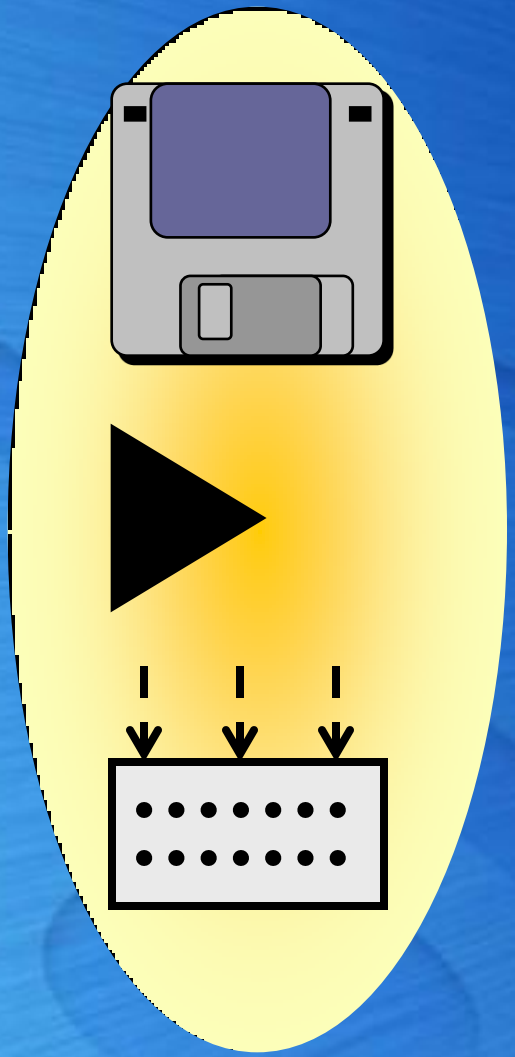
Propiedades	Configuración
(Name)	Textbox1
BackColor	Blue
Autosize	True
Visible	True
Border	Fixed 3D
Font	Microsoft SanSerif, 8.2 pt
Text	Textbox1

Cómo añadir código a los controles

- En la lista Nombre de clase, hacer clic en el control
- En la lista Nombre de método, hacer clic en el evento
- Añadir código entre Private Sub y End Sub

```
Private Sub Button1_Click(. . .)Handles
  Button1.Click
    'Colorar codigo aca
End Sub
```

Como guardar, ejecutar y generar la aplicación



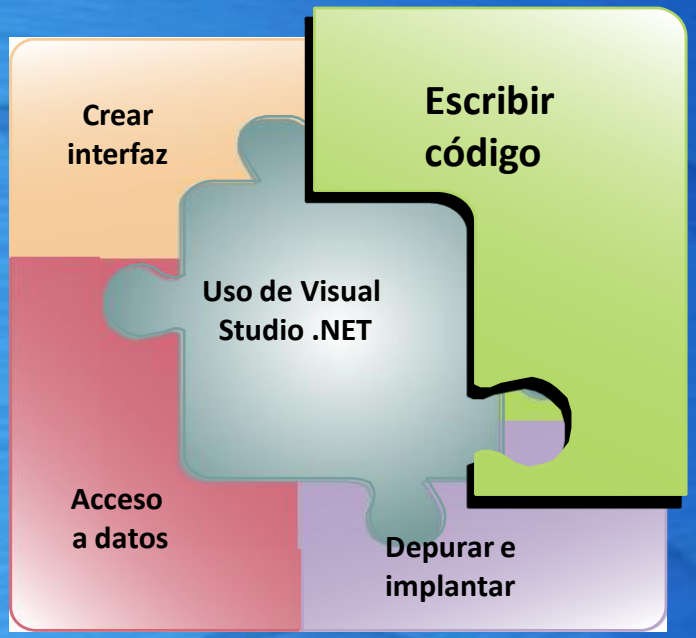
Guardar la aplicación

Ejecutar la solución en el entorno de desarrollo

Generar un archivo ejecutable

Elementos del lenguaje. Variables y estructuras de datos

Descripción



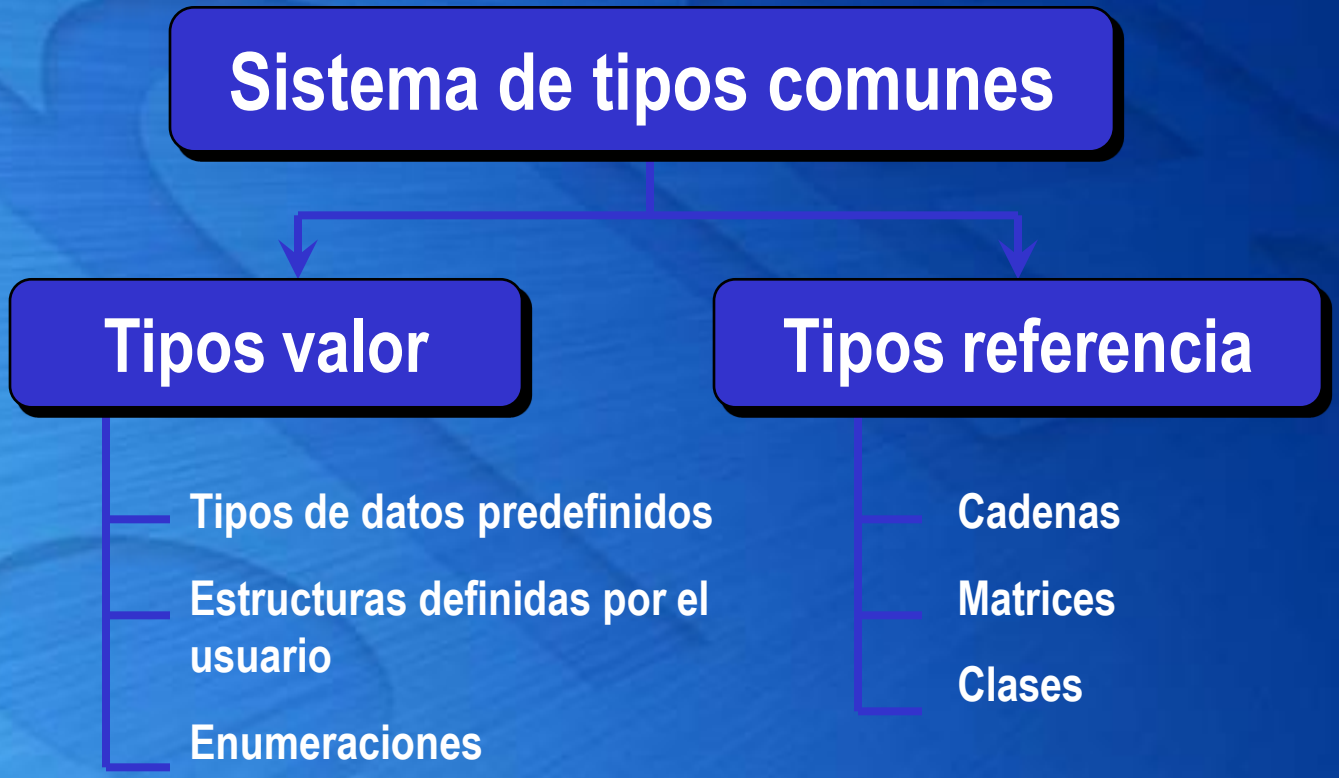
- Introducción a los tipos de datos
- Uso de variables
- Ámbito de las variables
- Convertir tipos de datos
- Crear y utilizar estructuras
- Almacenar datos en matrices

Introducción a los tipos de datos

- Sistema de tipos comunes
- Tipos valor
- Tipos referencia

¿Qué es el sistema de tipos comunes?

Define cómo funcionan los tipos en el Common Language Runtime



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 $7{,}9228 \times 10^{28}$ (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

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

Uso de variables

Tareas

1. Nombrar la variable
2. Declarar la variable
3. Asignar un valor a la variable
4. Utilizar la variable

¿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 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

Cómo nombrar 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
  - `NombreCliente` (PascalCasing)
  - `numeroCuenta` (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
```

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

Cómo asignar valores a las variables

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

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

- Asignar un valor a una variable mientras la declaramos

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

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

Variables frente a Constantes

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

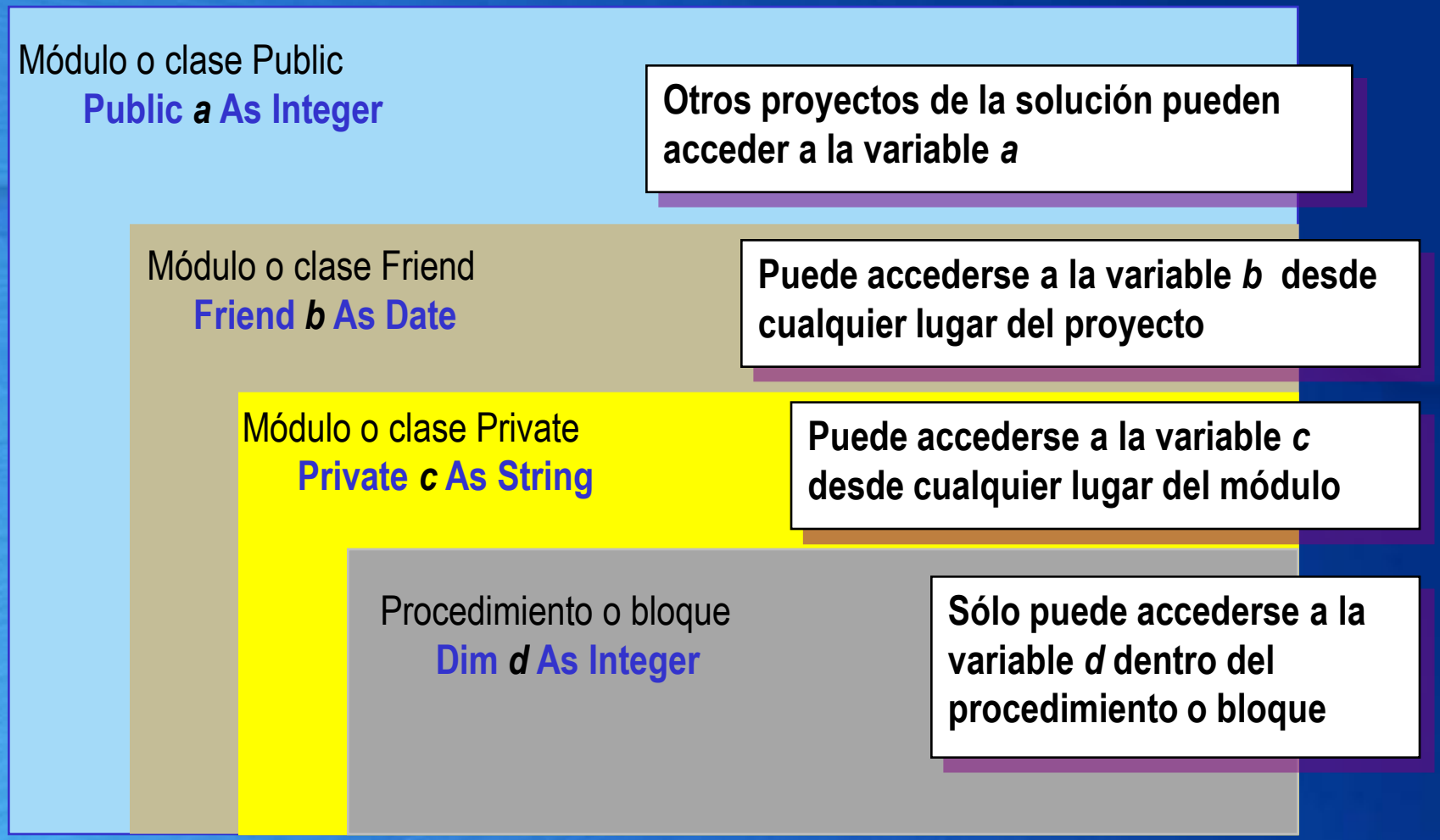
Sintaxis para declarar una constante:

```
Const constantName As Type
```

Práctica: Encontrar errores

1. Dim 12Count As Integer
2. Dim Number For Double
3. Const Son's Birthday As Day
4. Dim Error.Message As Text
5. Dim $CurrentExpenses With Decimal

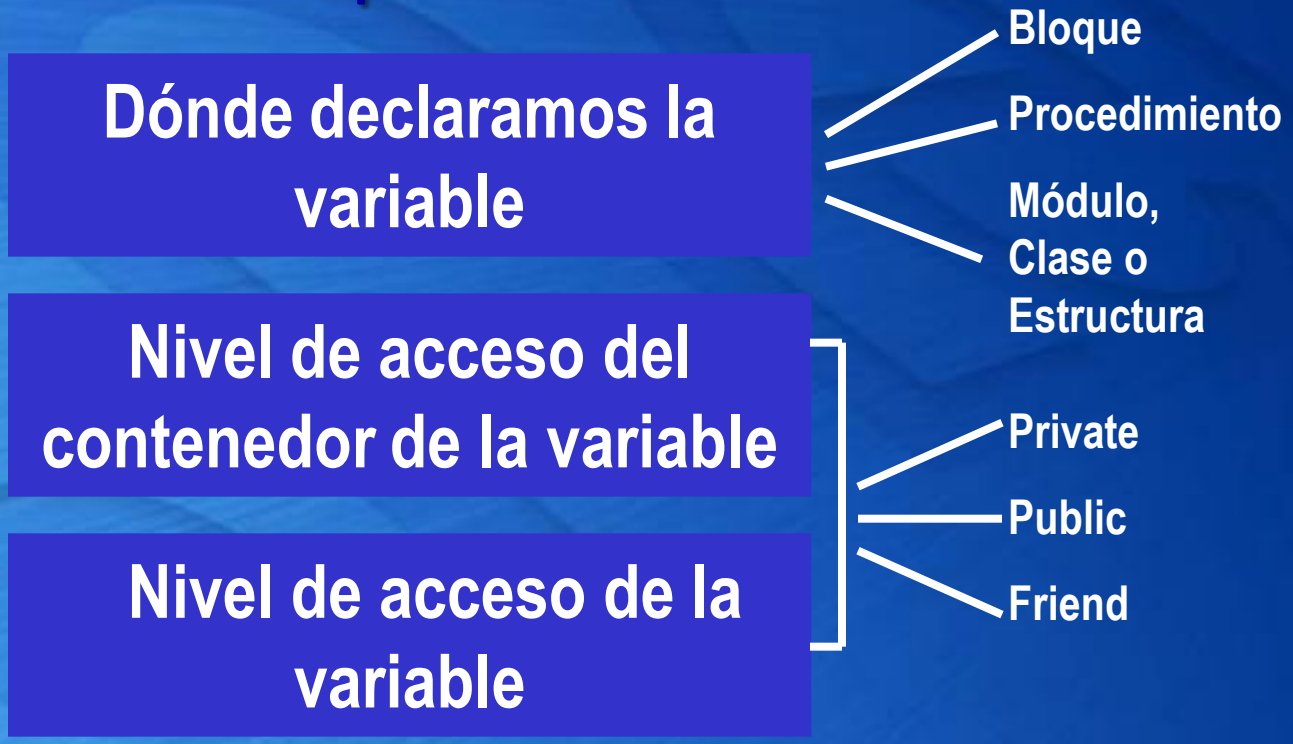
Ámbito de una variable



¿Qué es el ámbito?

Definición: ámbito es el conjunto de código al que se refiere una variable por su nombre

Factores que afectan al ámbito



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
```

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
```

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
```

Convertir tipos de datos

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

Funciones de conversión

Definición: las funciones de conversión permiten convertir valores de un tipo de datos a otro

Cómo convertir tipos de datos explícitamente

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)
```

Cómo funciona la conversión de datos implícita

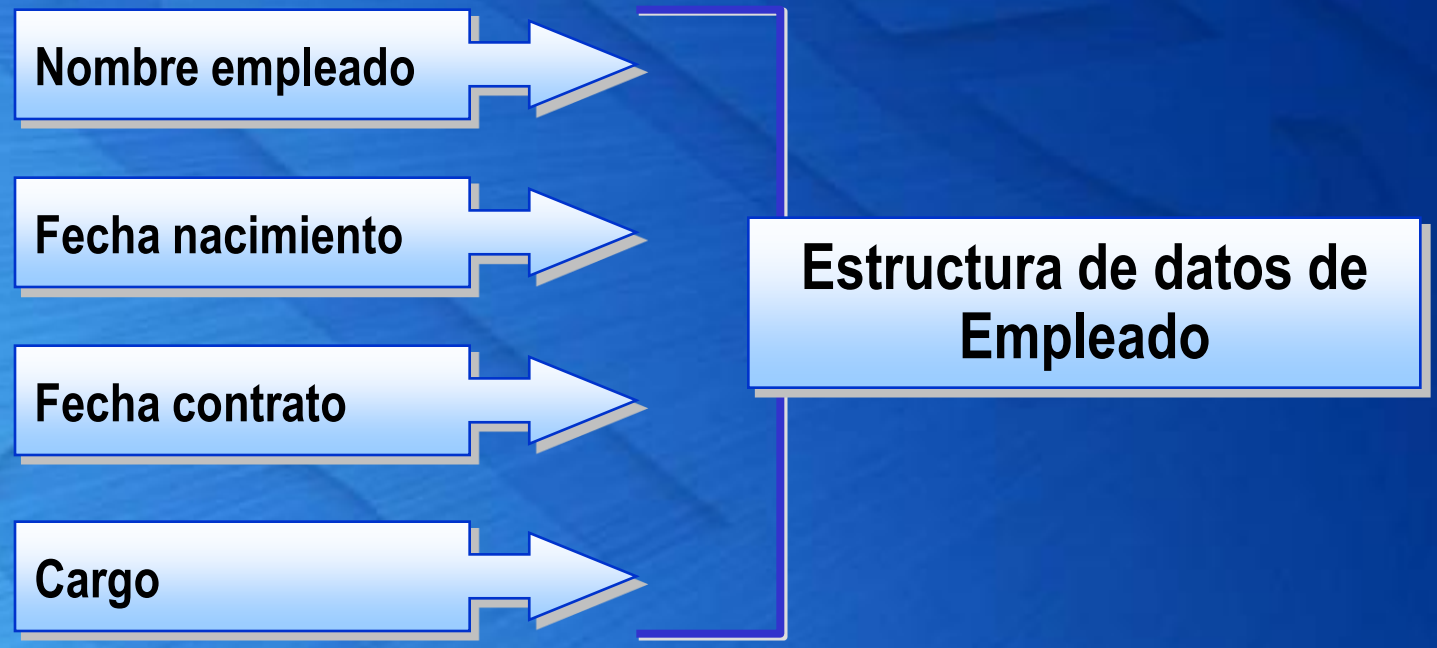
- Los tipos de datos son convertidos automáticamente
- No se requiere sintaxis especial en el código
- Ejemplo de conversión de datos implícita:

```
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

Crear y utilizar estructuras

Información relacionada de grupo → en una estructura única



¿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*

Cómo declarar estructuras

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

```
Modificador de Acceso Structure Nombre Estructura
    ' Declarar Miembros de Estructura
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

Práctica: Crear y utilizar estructuras

1. Declarar una estructura
2. Declarar una variable como tipo estructura
3. Asignar valores a los miembros de la estructura
4. Escribir código para utilizar los miembros de la estructura
5. Ejecutar y probar la aplicación

1.- Declarar una estructura

```
#Region "Declaracion de Estructura"
    Public Structure InfoCarro
        Dim Marca As String
        Dim Modelo As String
        Dim PrecioCompra As Single
        Dim FechaCompra As Date
    End Structure
#End Region
```

Estructura

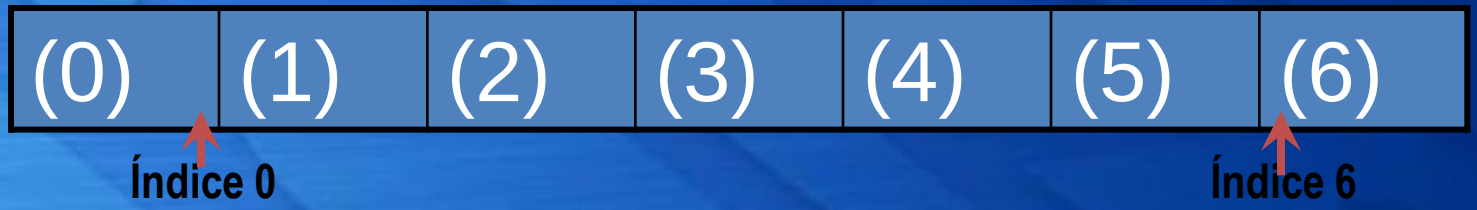
- 2. Declarar variable tipo estructura
  - Dim MiCarro As InfoCarro
- 3. Asignar valores a los miembros de la estructura
  - MiCarro.Marca = "Toyota"
  - MiCarro.Modelo = "Corolla"
  - MiCarro.PrecioCompra = 8000
  - MiCarro.FechaCompra = #1/1/2006#
- 4. Escribir código para utilizar los miembros de la estructura
  - Me.txtMarca.Text = MiCarro.Marca
  - Me.txtModelo.Text = MiCarro.Modelo
  - Me.txtPrecio.Text = MiCarro.PrecioCompra
  - Me.txtFecha.Text = MiCarro.FechaCompra
- 5. Ejecutar Programa

Almacenar datos en matrices

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

¿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



- Ejemplo
  - Para declarar una matriz entera con siete elementos:

```
Dim countHouses(7) As Integer
```

  - Para acceder al tercer elemento de la matriz:

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

Cómo declarar una matriz unidimensional

- Declaramos una matriz especificando el:
  - **Nombre de la matriz**
  - **Tamaño (número de elementos)**
  - **Tipo de datos de los elementos de la matriz**
  - **Modificador de acceso (si fuera necesario)**

```
AccessModifier ArrayName(Size) As Type
```

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
```

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:

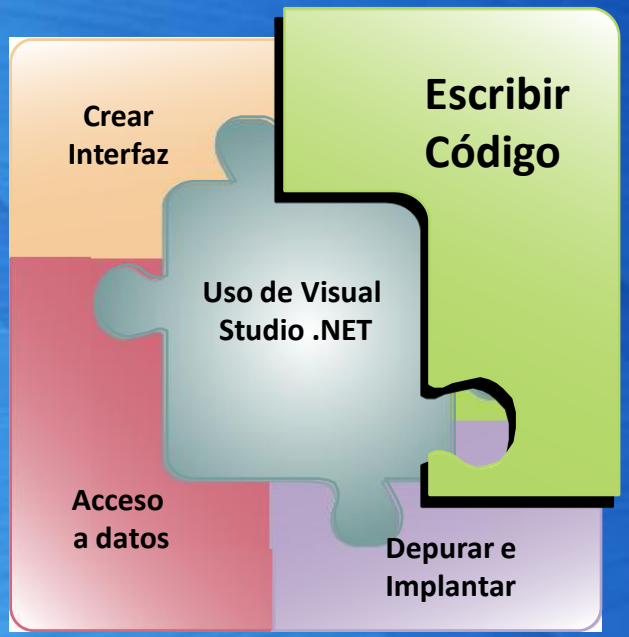
```
ReDim matrizExistente(NuevoTamaño)
```

- Ejemplo:

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

Funciones. Subrutinas y procedimientos

Descripción



- Crear procedimientos
- Uso de procedimientos
- Uso de funciones predefinidas

Crear procedimientos

- ¿Qué son los procedimientos?
- Cómo crear procedimientos Sub
- Cómo crear procedimientos Function
- Cómo declarar argumentos en procedimientos
- Cómo utilizar argumentos opcionales
- Reutilización del código

¿Qué son los procedimientos?

- Los procedimientos son las sentencias de código ejecutable de un programa, encerradas por una sentencia de declaración y una sentencia End
- Tres tipos:
  - Procedimientos **Sub** (incluyendo procedimientos **Sub** de eventos)
  - Procedimientos **Function**
  - Procedimientos **Property**
- Permitir la reutilización de código
- Declarados como *public* de forma predeterminada

Cómo crear procedimientos Sub

Los procedimientos Sub realizan acciones pero no devuelven un valor al procedimiento que realiza la llamada

```
[accessibility] Sub subname[(argumentlist)]
    ' Sub procedimiento statements
End Sub
```

Ejemplo:

```
Private Sub AboutHelp( )
   MessageBox.Show("MyProgram V1.0", "MyProgram Help")
End Sub
```

Cómo crear procedimientos Function

Los procedimientos Function realizan acciones y pueden devolver un valor al programa que realiza la llamada

```
[accessibility] Function name[(argumentlist)] As datatype
   ' Function statements, including optional Return
   ' statement
End Function
```

Ejemplo:

```
Public Function DoubleTheValue(ByVal J As Double) As _
   Double
  . . .
    Return J*2
  . . .
End Function
```

Cómo declarar argumentos en procedimientos

- Los *argumentos* son datos pasados a procedimientos
- Podemos pasar argumentos *ByVal* o *ByRef*
  - **ByVal:** El procedimiento no puede modificar el valor de la variable original
  - **ByRef:** El procedimiento puede modificar el valor de la variable original
  - **Excepción:** Los elementos no variables no se modifican en el código que llama, aunque sean pasados por referencia
- ByVal es el valor predeterminado en Visual Basic .NET
- Sintaxis y ejemplo:

```
([ByVal|ByRef] argumentname As datatype)
```

```
(ByVal Name As String)
```

Cómo utilizar argumentos opcionales

- Reglas para declarar argumentos opcionales:
  - Especificar un valor predeterminado
  - El valor predeterminado debe ser una expresión constante
  - Los argumentos que sigan a un argumento opcional también deben ser opcionales
- Sintaxis:

```
(Optional [ByVal|ByRef] argumentname As datatype = defaultvalue)
```

- Ejemplo:

```
Function Add (ByVal value1 As Integer, ByVal value2 As _
   Integer, Optional ByVal value3 As Integer = 0) As Integer
```

Reutilización del código

Usar…	para…	Ejemplos
Estructura	Objetos que no necesitan ser extendidos	**Size** **Point**
Módulo	Funciones de utilidad y datos globales	**Conversión de temperatura**
Clase	Extende objetos u objetos que necesitan cleanup	**Formularios** **Botones**

- Crear un módulo:

```
[Public|Friend] Module ModuleName
End Module
```

Práctica: Crear una función en un módulo

1. Abrir un proyecto
2. Añadir un módulo al proyecto
3. Crear una función en el módulo
4. Escribir el código para la función

Uso de procedimientos

- Cómo utilizar procedimientos Sub
- Cómo utilizar procedimientos Function
- Cómo pasar matrices a procedimientos
- Cómo crear un Sub Main

Cómo utilizar los procedimientos Sub

```
Public Sub Hello(ByVal name As String)
  MessageBox.Show("Hello " & name)
End Sub
```

```
Sub Test( )
  Hello("John")
End Sub
```

Cómo utilizar los procedimientos Function

- Invocar una función
  - Incluir el nombre de la función y los argumentos en el lado derecho de una instrucción de asignación

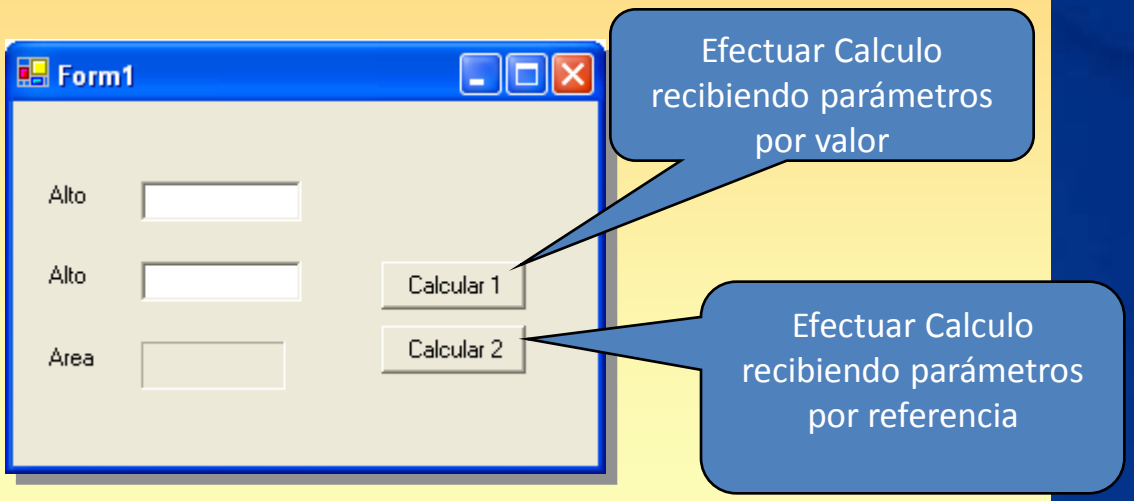
```
Dim celsiusTemperature As Single
celsiusTemperature = FtoC(80)
```

  - Utilizar el nombre de la función en una expresión

```
If  FtoC(userValue) < 0 Then ...
End If
```

Práctica: utilización del valor devuelto de una función

1. **Crear el interfaz de usuario**
2. **Escribir código para la aplicación**



Funciones dentro del modulo

```
Function AreaPorValor(ByVal alto As Single, ByVal ancho As
    Single) As Single
        AreaPorValor = alto * ancho
End Function
Function AreaPorReferencia(ByRef alto As Single, ByRef ancho
    As Single) As Single
        alto *= 2
        ancho *= 2
        AreaPorReferencia = alto * ancho
End Function
```

Eventos Click de los botones

```
Private Sub cmdCalcular1_Click(ByVal sender As System.Object, ByVal e As
    System.EventArgs) Handles cmdCalcular1.Click
      lblAreaResultado.Text = AreaPorValor(CInt(txtAlto.Text),
    CInt(txtAncho.Text))
End Sub

Private Sub cmdCalcular2_Click(ByVal sender As System.Object, ByVal e As
    System.EventArgs) Handles cmdCalcular2.Click
      Dim iAlto As Single = CInt(txtAlto.Text)
      Dim iAncho As Single = CInt(txtAncho.Text)
      lblAreaResultado.Text = AreaPorReferencia(iAlto, iAncho)
      txtAlto.Text = iAlto
      txtAncho.Text = iAncho
      txtAlto.Refresh()
      txtAncho.Refresh()
End Sub
```

Cómo pasar matrices a procedimientos

- Una matriz se pasa igual que otros argumentos:

```
Sub PassArray(ByVal testScores As Integer( ))
End Sub

Dim scores( ) As Integer = {80, 92, 73}
PassArray(scores)
```

- Declarar una matriz de parámetros:

```
Sub StudentScores(ByVal name As String, ByVal _
        ParamArray scores( ) As String)
    ' Statements for Sub procedure
End Sub
```

- Invocar un procedimiento con una matriz de parámetros:

```
StudentScores("Anne","10","26","32","15","22","16")
```

Cómo crear un Sub Main

- Sub Main: Punto de inicio de la aplicación
- Application.Run: Inicia la aplicación
- Application.Exit: Cierra la aplicación

Práctica: Crear un Sub Main

1. Declarar variables a nivel de módulo
2. Crear un procedimiento Sub Main y establecerlo como el objeto de inicio
3. Escribir código para el formulario Principal
4. Escribir código para cerrar la aplicación
5. Probar la aplicación

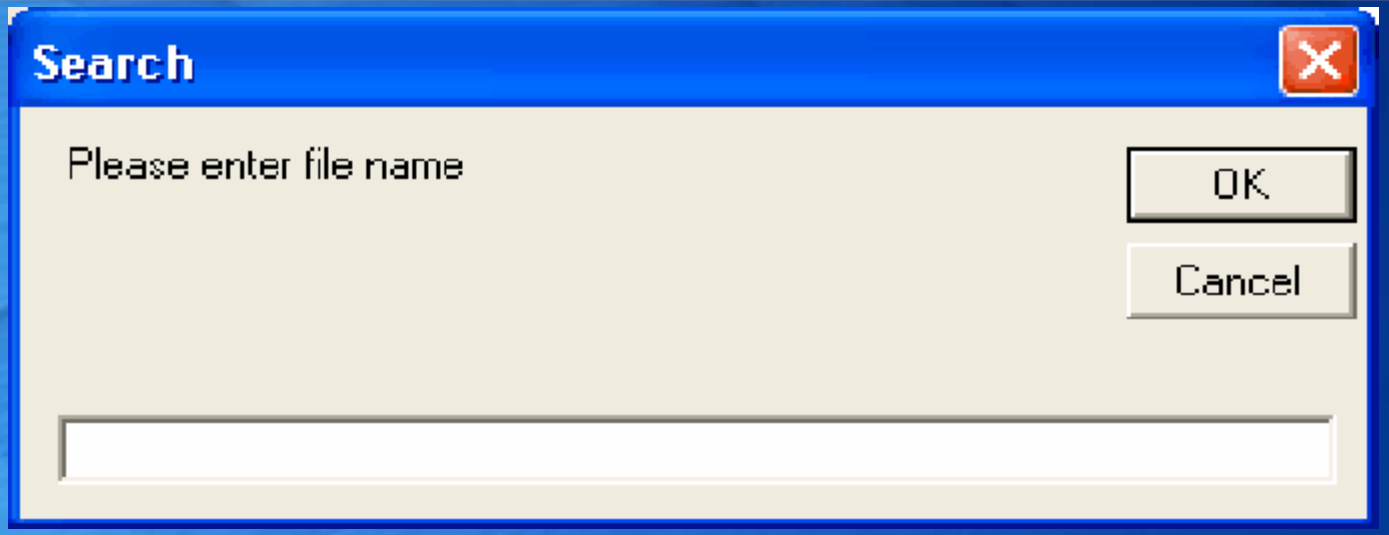
Lección: Uso de funciones predefinidas

- Cómo utilizar la función InputBox
- Cómo utilizar las funciones de fecha y hora
- Cómo utilizar las funciones String
- Cómo utilizar las funciones Format
- Cómo utilizar las funciones Financial

Cómo utilizar la función InputBox

- Muestra un mensaje en un cuadro de diálogo y devuelve al usuario la entrada en una cadena

```
Dim FileName As String
FileName = InputBox("Please enter file name","Search")
```



Cómo utilizar las funciones de fecha y hora

- Realizan cálculos y operaciones que implican fechas y horas
- Ejemplos:
  - **DateAdd**: Añade o sustrae un intervalo de tiempo específico a una fecha

```
DateAdd(DateInterval.Day, 10, billDate)
```

  - **DateDiff**: Determina cuántos intervalos de tiempo especificados existen entre dos valores de fecha/hora

```
DateDiff(DateInterval.Day, Now, secondDate)
```

Cómo utilizar las funciones String

- Extraen sólo una parte determinada de una cadena
- Devuelven información sobre una cadena
- Muestran información de un formato determinado
- Ejemplos:
  - **Trim**

```
NewString = Trim(MyString)
```

  - **Len**

```
Length = Len(customerName)
```

  - **Left**

```
Microsoft.VisualBasic.Left(customerName, 5)
```

Cómo utilizar las funciones Format

- Formatean números, fechas y horas según estándares aceptados
- Muestran formatos regionales sin codificar de nuevo para nacionalidades o regiones
- Ejemplos:
  - **FormatCurrency**

```
FormatCurrency(amountOwed, , , TriState.True,TriState.True)
```

  - **FormatDateTime**

```
FormatDateTime(myDate, DateFormat.LongDate)
```

Cómo utilizar las funciones Financial

- Realizan cálculos y operaciones que implican finanzas; por ejemplo, tipos de interés
- Ejemplos:
  - **Pmt**

```
payment = Pmt(0.0083, 24, -5000, 0, DueDate.BegOfPeriod)
```

  - **Rate**

```
ratePerPeriod = Rate(24, 228, -5000, 0, DueDate.BegOfPeriod, _
   0.8)*100
```