# 3. Sintaxis de C#

## 3.1 Variables, constantes y enumeraciones

3.1.1 **LAS VARIABLES:** Son espacios de memoria dentro del computador que permiten almacenar, durante la ejecución de su aplicación, diferentes valores útiles para el funcionamiento de su programa. Se debe declarar una variable obligatoriamente antes de su uso en el código.

Algunas consideraciones importantes:

- El nombre de una variable debe empezar con una letra.
- Puede estas formada por letras, cifras o por el carácter de subrayado.
- Puede contener un máximo de 1,023 caracteres.
- C# distingue entre mayúsculas y minúsculas.
- No se deben usar las palabras reservadas del lenguaje.

3.1.1.1 **Tipos de Variables**: al especificar el tipo de una variable, indicamos qué datos vamos a poder almacenar en ella.

Hay dos categorías de tipos de variables disponibles:

- Los tipos por valor: la variable contiene realmente los datos.
- Los tipos referencia: la variable contiene la dirección de la memoria donde se encuentran los datos.

Estas a su vez se pueden clasificar en seis tipos:

**a. Tipos Numéricos enteros:**

| Tipos enteros con signo | | | |
|---|---|---|---|
| sbyte | -128 | 127 | 8 bits |
| short | -32768 | 32767 | 16 bits |
| int | -2147483648 | 2147483647 | 32 bits |
| long | -9223372036854775808 | 9223372036854775807 | 64 bits |

| Tipos enteros sin signo | | |
|---|---|---|
| byte | 255 | 8 bits |
| ushort | 65535 | 16 bits |
| uint | 4294967295 | 32 bits |
| ulong | 18446744073709551615 | 64 bits |

**b. Tipos decimales:**

| | | | |
|---|---|---|---|
| float | 3.40282347E+38 | 3.40282347E+38 | 4 bytes |
| double | 1.7976931348623157E+308 | 1.7976931348623157E+308 | 8 bytes |
| decimal | 79228162514264337593543950335 | 79228162514264337593543950335 | 16 bytes |

**c. Tipos caracteres:**

Representa una serie de cero a 2.147.483.648 caracteres. Si algunos caracteres especiales deben aparecer en una cadena, se deben especificar con una secuencia de escape.

| Secuencia de escape | Carácter |
|---|---|
| \' | Comilla simple |
| \" | Comilla doble |
| \\ | Barra inversa |
| \0 | Carácter nulo |
| \a | Alerta |
| \b | Backspace |
| \f | Salto de página |
| \n | Salto de línea |
| \r | Retorno de carro |
| \t | Tabulación horizontal |
| \v | Tabulación vertical |

Ejemplos:

```
String Nombre = "Juan";
String Apellido = "Cuevas";
String NombreCompleto = Nombre + "" + Apellido;
```

Las dos sentencias siguientes hacen lo mismo:

```
String cadena = "¿Qué dice? Él dice \"hola\"";
String cadena2 = @"¿Qué dice? Él dice ""hola""";
```

**d. Tipo bool:**

El tipo bool permite utilizar una variable que puede tener dos estados verdadero/falso. La asignación se hace directamente con los valores true o false, como en el ejemplo siguiente:

Ejemplos:

```
bool Disponible=true;
bool Modificable = false;
```

**e. El tipo Object**

Este es el tipo más universal de Visual C#. En una variable de tipo Object, usted puede almacenar cualquier cosa. En realidad, este tipo de variable no contendrá el propio valor, sino la dirección, en la memoria de la máquina, donde se podrá encontrar el valor de la variable.

**f. Tipo Nullable**

Permiten a las variables de tipo valor no contener ninguna información. Consiste en utilizar un valor que no está bien definido. Para activar esta funcionalidad en una variable sólo hay que utilizar el carácter "?" después del tipo de la variable. Antes de utilizar una variable de este tipo hay que verificar si contiene realmente un valor. Por ejemplo:

```
int? CodigoPostal;
CodigoPostal=17000;
```

```
if (CodigoPostal.HasValue)
{
        Console.WriteLine(CodigoPostal.Value);
}
else
{
        Console.WriteLine("Código postal vacío");
}
```

#### 3.1.1.2 Conversiones de tipos:

Las conversiones de tipos consisten en transformar una variable de un tipo en otro tipo. Las conversiones se pueden efectuar hacia un tipo superior o inferior.

Si se convierte hacia un tipo inferior, hay riesgo de pérdida de información. Por ejemplo, la conversión de un tipo double hacia un tipo long hará perder la parte decimal del valor:

Ejemplo:

```
double x;
long y;
x = 21.2343433333333;
y = x;
```

Este tipo de conversión no está totalmente prohibido. Sólo tiene que avisar al compilador de su intención utilizando una operación de conversión explícita:

```
double x;
long y;
x = 21.2343433333333;
y = (long)x;
```

También es posible convertir un valor a otro utilizando la clase Convert. Esta tiene una diversidad de métodos de conversión. Ejemplo:

```
double x;
long y;
x = 21.2343433333333;
y = Convert.ToInt32(x);
```

#### 3.1.1.3 Formato de valores numéricos:

**Currency:** Formato monetario tal como está definido en las opciones regionales y lingüísticas del panel de configuración del sistema. Ejemplo:

```
String.format("{0:c}",12.35);
Resultado: 12,35 €
```

**Percent:** Multiplica el valor indicado por cien y añade el símbolo « % » después. Ejemplo:

```
String.format("{0:p}",0.2);

Resultado: 20,00%
```

**Standard:** Formato numérico tal como está definido en las opciones regionales y lingüísticas del panel de configuración del sistema. *Ejemplo:*

```
String.format("{0:n}",245813.5862);

Resultado: 245.813,59
```

**Cadena de formateo personalizada para valores numéricos:**

Reserva una ubicación para un carácter numérico. Los ceros no significativos se visualizan. *Ejemplo:*

```
String.Format("{0:00000000000.0000}",245813.12)
```

```
Resultado: 00000245813,1200
```

### 3.1.1.4 Formato de fecha y hora:

**G**

Formato Fecha corta y formato Hora tal como está definido en las opciones regionales y lingüísticas del panel de configuración del sistema. *Ejemplo:*

```
String.Format("{0:G}", DateTime.Now);
```

```
Resultado 25/03/2008 11:10:42
```

**D**

Formato Fecha larga tal como está definido en las opciones regionales y lingüísticas del panel de configuración del sistema. *Ejemplo:*

```
String.Format("{0:D}", DateTime.Now);
```

```
Resultado martes 25 marzo del 2008
```

**d**

Formato Fecha corta tal como está definido en las opciones regionales y lingüísticas del panel de configuración del sistema. *Ejemplo:*

```
String.Format("{0:d}", DateTime.Now);
```

```
Resultado 25/03/2008
```

### 3.1.1.5 Declaración de variables:

El compilador considera que toda variable que aparece en una aplicación debe haber sido declarada.

La sintaxis general de declaración de una variable es la siguiente:

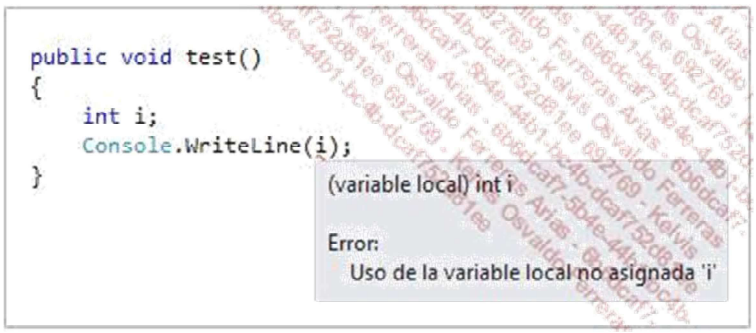
```
Tipo de la variable nombreVariable [=valor inicial][,nombreVariable2]
```

Los parámetros entre corchetes son opcionales.

En caso de que se omita el valor inicial, la variable será inicializada a cero si corresponde a un tipo **numérico**; a una cadena de carácter vacío si es del tipo **String**; al valor null si es del tipo **Object**, y a false si es del tipo **bool**.

**Estas reglas no se aplican a las variables declaradas en el interior de una función que deben ser inicializadas antes de poder utilizarse.** Esta inicialización puede ocurrir en el momento de la declaración o con posterioridad, pero obligatoriamente antes de que una instrucción utilice el contenido de la variable.

*Ejemplo:*




### 3.1.1.6 Inferencia de tipo

Gracias a la inferencia de tipo, el compilador puede determinar el tipo que se ha utilizar para una variable local. Para ello, se basa en el tipo de la expresión utilizada para inicializar la variable. El nombre de la variable debe venir precedido en este caso de la palabra reservada `var`. En el ejemplo siguiente la variable se considera como una cadena de caracteres:

```
var apellido = "García";
```

Para que la inferencia de tipo funcione correctamente, es imperativo respetar algunas reglas:

- La inicialización debe hacerse en la misma línea de código que la declaración.
- La inferencia sólo funciona para las variables locales, es decir, las declaradas en una función.

### 3.1.1.7 Ambito de las variables:

El ámbito de una variable es la porción de código en la cual se puede trabajar con dicha variable. Depende de la ubicación donde se sitúa la declaración y de la palabra reservada utilizada para la declaración.

**Ambito a nivel de bloque:**

Sólo el código del bloque tendrá la posibilidad de trabajar con la variable (por ejemplo, en un bucle `for next`).

**Ambito a nivel de función:**

Sólo el código de la función donde se declara la variable podrá modificar su contenido. Este tipo de variable se llama a veces «variable local».

**Ambito a nivel de clase:**

Una variable declarada en el interior de una clase es accesible al código de esta clase sin restricción y eventualmente a partir de otras porciones de código en función del nivel de acceso de la variable. Sin embargo, una instancia de la clase deberá estar disponible obligatoriamente para que la variable sea accesible. El caso particular de las variables estáticas (también llamadas variables de clase) se tratará en el capítulo dedicado a la programación orientada a objetos.

### 3.1.1.8 Nivel de acceso de las variables:

El nivel de acceso de una variable se combina con el ámbito de la variable y determina qué porción de código puede leer y escribir en la variable. Un conjunto de palabras reservadas permite controlar el nivel de acceso. Se utilizan al principio de la instrucción de declaración de la variable únicamente para las variables declaradas en una clase.

**Public:**

Los elementos declarados con la palabra reservada `public` serán accesibles desde cualquiera porción de código
del proyecto en el cual están declarados y desde cualquier otro proyecto que haga referencia a aquel donde están declarados. La palabra reservada `public` no se puede utilizar, sin embargo, para la declaración en el interior de una función.

**Protected:**

Esta palabra reservada se puede utilizar en el interior de una clase. Permite restringir el acceso a la variable, al código de la clase y al código de todas las clases que heredan de ella.

**Internal:**

Los elementos declarados con esta palabra reservada serán accesibles desde el ensamblado en el cual están declarados. Esta palabra reservada no se puede utilizar en el interior de una función.

**Protected Internal:**

Este nivel de acceso es la unión de los niveles de acceso protected e internal. Hace visible la variable al conjunto del ensamble en el cual está declarada y a todas las clases que heredan de aquella donde está declarada.

**Private:**

Esta palabra reservada restringe el acceso a la variable al módulo, a la clase o a la estructura en la cual está declarada. No se puede utilizar en el interior de un procedimiento o función. Si no se utiliza ninguna palabra reservada durante la declaración de la variable, se considera como private. Sin embargo es preferible siempre especificar un nivel de acceso para una variable.

#### 3.1.1.9 Duración de vida de las variables

La duración de vida de una variable nos permite especificar durante cuánto tiempo, mientras se ejecuta nuestra aplicación, el contenido de nuestra variable estará disponible. Para una variable declarada en un procedimiento o una función, la duración de vida corresponde a la duración de ejecución del procedimiento o de la función. En el momento en que finalice la ejecución del procedimiento o función se elimina la variable de la memoria. Se volverá a crear durante la próxima llamada del procedimiento o función. La duración de vida de las variables de nivel de clase se asocia a la duración de vida de las instancias de la clase.

### 3.1.2 LAS CONSTANTES:

Es un espacio de memoria dentro del computador que permite almacenar un valor que no puede ser cambiado durante la ejecución del programa. La definición de una constante se efectúa por la palabra reservada const. Ejemplo:

```
const int ValorMax = 100;
const double impuesto = 0.18;
const string Mensaje = "Demasiado alto";
```

El valor de una constante se puede también calcular desde otra constante. Ejemplo:

```
const int Total = 100;
const int Semi = Total / 2;
```

### 3.1.3 LAS ENUMERACIONES:

Una enumeración nos va a permitir definir un conjunto de constantes que están vinculadas entre sí. La declaración se efectúa de la manera siguiente:

```
enum dias
    {

        Domingo,
        Lunes,
        Martes,
        Miércoles,
        Jueves,
        Viernes,
        Sábado
```

```
}
```

Por defecto, el primer valor de la enumeración se inicializa a cero. Luego las constantes siguientes se inicializan con un incremento de uno. La declaración anterior hubiera podido escribirse:

```
const int Domingo = 0;
const int Lunes = 1;
const int Martes = 2;
const int Miércoles = 3;
const int Jueves = 4;
const int Viernes = 5;
const int Sábado = 6;
```

Los tipos admitidos para una enumeración son **byte**, sbyte, short, ushort, int, uint, long o ulong. Ejemplo:

```
enum dias : byte {Sat=1, Sun, Mon, Tue, Wed, Thu, Fri};
```

## *3.2 Los operadores*

Los operadores son palabras reservadas del lenguaje que permiten la ejecución de operaciones en el contenido de ciertos elementos, en general variables, constantes, valores literales o devoluciones de funciones. La combinación de uno o varios operadores y elementos en los cuales los operadores van a apoyarse se llama una expresión. Estas expresiones se valoran en el momento de su ejecución, en función de los operadores y valores que son asociados.

Los operadores se pueden clasificar en:

### 3.2.1 Los operadores de asignación

El único operador disponible en esta categoría es el operador =. Permite asignar un valor a una variable. Se usa siempre el mismo operador, sea cual sea el tipo de variable (numérico, cadena de caracteres...).

### 3.2.2 2. Los operadores aritméticos

Los operadores aritméticos permiten efectuar cálculos en el contenido de las variables:

| Operador | Operación realizada | Ejemplo | Resultado |
|---|---|---|---|
| + | Suma | 6+4 | 10 |
| - | Sustracción | 12-6 | 6 |
| * | Multiplicación | 3*4 | 12 |
| / | División | 25/3 | 8.3333333333 |
| % | Módulo (resto de la división entera) | 25 % 3 | 1 |

### 3.2.3 Los operadores de comparación

Los operadores de comparación se utilizan en las estructuras de control de una aplicación (if, do loop...). Devuelven un valor de tipo boolean en función del resultado de la comparación efectuada. Luego este valor será utilizado por la estructura de control.

| Operador | Operación realizada | Ejemplo | Resultado |
|---|---|---|---|
| == | Igualdad | 2 == 5 | False |
| != | Desigualdad | 2 != 5 | True |
| < | Inferior | 2 < 5 | True |
| > | Superior | 2 > 5 | False |
| <= | Inferior o igual | 2 <= 5 | True |
| >= | Superior o igual | 2 >= 5 | False |

| | | | |
|---|---|---|---|
| is | Comparación del tipo de la variable con el tipo dado | O1 is Cliente | True si la variable O1 referencia un objeto creado a partir del tipo cliente |

### 3.2.4 Operador de concatenación

El operador se utiliza para la concatenación de cadenas de caracteres. Es el mismo operador que se utiliza para la suma. Sin embargo, no hay riesgo de confusión, ya que Visual C# no hace conversión implícita de las cadenas de caracteres en numérico. Determina por lo tanto que, si uno de los dos operandos es una cadena de caracteres, se debe ejecutar una concatenación, incluso si una de las cadenas representa un valor numérico.

El código siguiente:

```
string cadena = "123";
Console.WriteLine(cadena + 456);
```

Visualiza: 123456

### 3.2.5 Los operadores lógicos

Los operadores lógicos permiten combinar las expresiones en estructuras condicionales o de bucle.

| Operador | Operación | Ejemplo | Resultado |
|---|---|---|---|
| & | Y lógico | If (test1) & (test2) | verdadero si test1 **y** test2 es verdadero |
| \| | O lógico | If (test1) \| (test2) | verdadero si test1 **o** test2 es verdadero |
| ^ | O exclusivo | If (test1) ^ (test2) | verdadero si test1 o test2 es verdadero, pero no si los dos son verdaderos simultáneamente |
| ! | Negación | If Not test | Invierte el resultado del test |
| && | Y lógico | If (test1) && (test2) | Idem «y lógico» pero test2 sólo será evaluado si test1 es verdadero |
| \|\| | O lógico | If (test1) \|\| (test2) | Idem «o lógico» pero test2 sólo será evaluado si test1 es falso |

Conviene ser prudente con los operadores && y || ya que la expresión que prueba en segundo término (test2 en nuestro caso) puede no llegar a ser ejecutada. Si esta segunda expresión modifica una variable, ésta se modificará sólo en los siguientes casos:

primer test verdadero en el caso del &&,

primer test falso en el caso del ||.

### 3.2.6 Orden de evaluación de los operadores

Cuando varios operadores se combinan en una expresión, son valorados en un orden muy preciso. En primer lugar se resuelven las operaciones aritméticas, luego las operaciones de comparación y entonces los operadores lógicos.

Los operadores aritméticos tienen también entre ellos un orden de evaluación en una expresión. El orden de evaluación es el siguiente:

- Negación (-)
- Multiplicación y división (*, /)
- Módulo (%)
- Suma y sustracción (+, -), concatenación de cadenas (+)

Si necesita un orden de evaluación diferente en su código, dé prioridad a las porciones que se han de evaluar primero colocándolas entre paréntesis, como en la siguiente expresión:

```
X= (z * 4) + (y * (a + 2));
```

Nota: Usted puede utilizar tantos niveles de paréntesis como desee en una expresión. Es importante, sin embargo, que la expresión contenga tantos paréntesis cerrados como paréntesis abiertos; si no el compilador generará un error.