

# Introducción a la Programación con C# .NET

Instructor : Miriam Myszne

Email : [mmyszne@gmail.com](mailto:mmyszne@gmail.com)

# Intro. a la Programación con C# .NET

## Contenido del módulo

- Introducción a .NET y al lenguaje de programación C#
- Sus orígenes
- Uso de variables en C#
- Tipos de datos en C#
- Uso de los diferentes Operadores de C#
- Uso de Constantes
- Uso de las diferentes Sentencias de C#
- Uso de Comentarios
- Ventajas y diferencias entre tipos de comentarios
- Anexo

# Intro. a la Programación con C# .NET

## Qué es .NET

Es una plataforma de desarrollo multi-lenguaje, pensada para correr en distintos entornos operativos, y unifica distintos tipos de aplicaciones a crear (aplicaciones Windows, de consola, WEB, etc.) en un mismo ambiente de trabajo.

.NET está formado por cuatro componentes principales:

- El **CLR** (Common Language Runtime)
- El **MSIL** (Microsoft Intermediate Language)
- La **biblioteca de clases** .NET Framework
- El entorno de programación (**IDE**)

# Intro. a la Programación con C# .NET

## CLR (Common Language Runtime)

Es el entorno de ejecución para las aplicaciones escritas en cualquier lenguaje de .NET (C#, Visual Basic, C++, etc.).

## MSIL (Microsoft Intermediate Language)

Representa el código objeto al que es transformado el código fuente escrito en cualquier lenguajes de .NET.

El código intermedio luego será compilado a código nativo por el compilador **JIT (Just In Time)** que provee la CLR en tiempo de ejecución.

# Intro. a la Programación con C# .NET

## Biblioteca de clases .NET Framework

Es una extensa colección de clases base, para todos los propósitos imaginables, usada por todos los lenguajes e IDEs que corren sobre .NET.

## Entorno de programación (IDE)

Es el ambiente de trabajo que tiene herramientas de diseño visual, editores, debuggers y diversos asistentes, para que el desarrollador pueda diseñar, escribir y probar sus aplicaciones. Existen varios IDEs (SharpDevelop, RAD Studio, etc.).

Nosotros usaremos el propio de Microsoft: **Visual Studio**.

# Intro. a la Programación con C# .NET

## Qué es C# (C Sharp)

C# es un lenguaje que pretende reunir lo mejor de los diversos lenguajes que compilan a código nativo (C/C++, Object Pascal, ADA, etc.), y de aquellos interpretados (Java, Perl, etc.) en uno solo, y además, pueda correr sobre diversos sistemas operativos.

C# toma gran parte de la sintaxis de C/C++, y muchas de las características del lenguaje Eiffel, poniendo a disposición del desarrollador un potente lenguaje 100% orientado a objetos.

El framework .NET está 100% escrito en C# ; por lo tanto ES EL LENGUAJE BASE PARA .NET.

# Intro. a la Programación con C# .NET

## El compilador

Cuando se instala en cada computadora el SDK de MS.NET, se crea la carpeta **c:\WINDOWS\Microsoft.NET\Framework\v4.0.30319\** según la versión del framework; allí están los compiladores de los lenguajes de .NET.

El compilador para C# es **csc.exe** y puede ser invocado desde la línea de comandos (Inicio | Ejecutar | cmd) además de usando el IDE.

Invocando al compilador y pasándole como parámetro un archivo **.cs**, se obtiene el **\*.exe** (ejecutable). Este ejecutable no es autónomo, necesita del framework .NET con el CLR instalado en la misma máquina, para poder correr.

# Intro. a la Programación con C# .NET

## Los archivos fuentes \*.cs

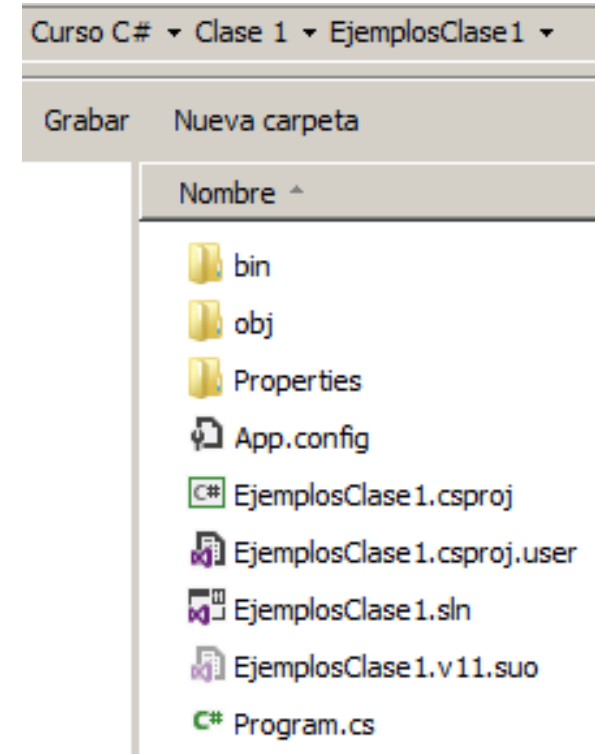
Los archivos de “**código fuente**” no son más que archivos de texto plano, que contienen las líneas del programa en C#, con las instrucciones. Deben cumplir con las normas sintácticas del lenguaje, caso contrario, el compilador detecta los errores y no genera el ejecutable.

**Para realizar el ejemplo siga los siguientes pasos junto al instructor:**



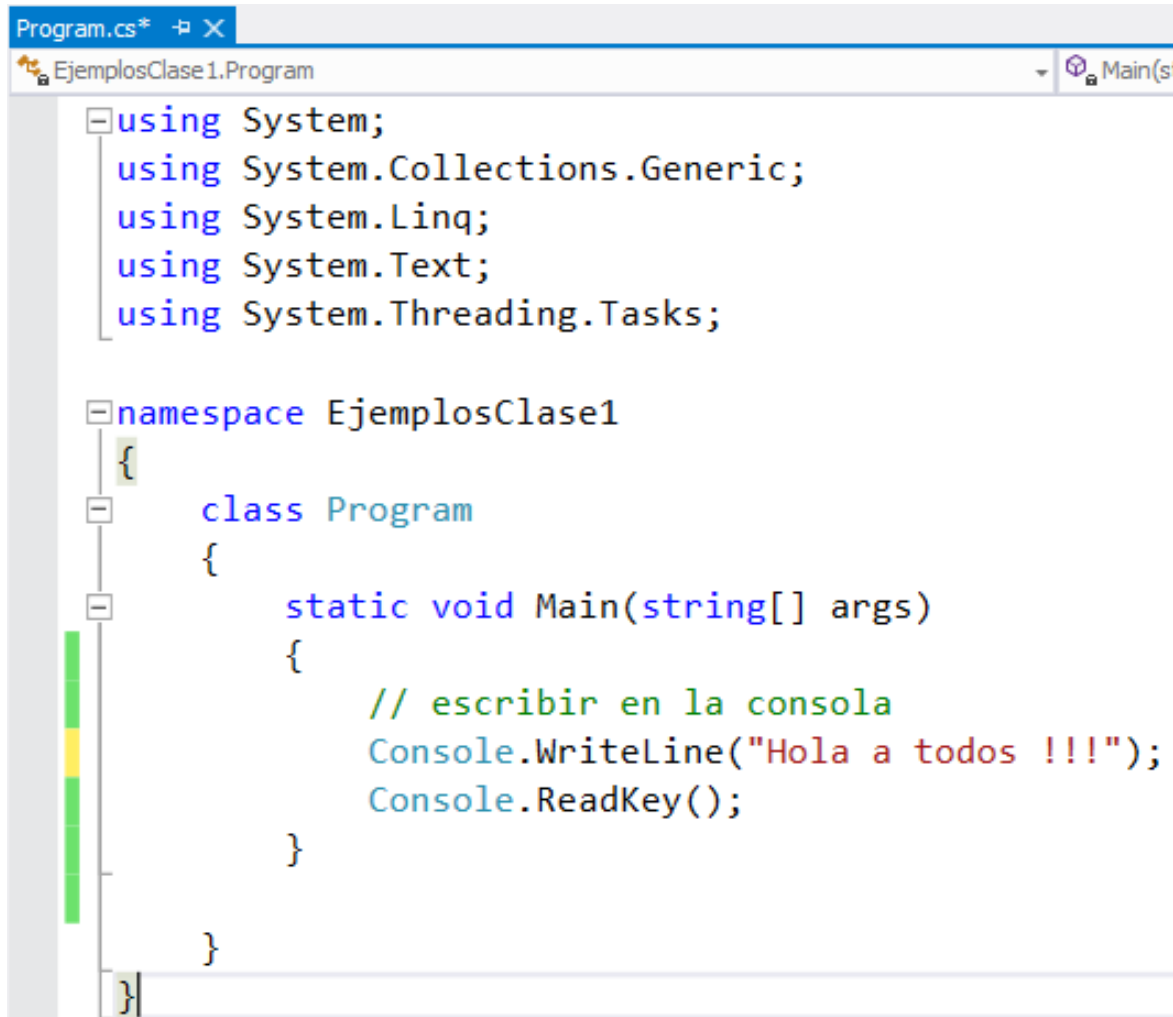
# Intro. a la Programación con C# .NET

- Crear una carpeta para guardar los archivos del curso
- Iniciar Visual Studio
- Seleccionar **menú Archivo – Nuevo Proyecto**
- Seleccionar **Visual C# - Aplicación de consola**
- Nombrar al proyecto como **EjemplosClase1** y aceptar
- Seleccionar **menú Archivo - Guardar todo** y seleccionar la carpeta creada



# Intro. a la Programación con C# .NET

Escribir en la consola un saludo y ejecutar:



```
Program.cs*  EjemplosClase1.Program  Main(s
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace EjemplosClase1
{
    class Program
    {
        static void Main(string[] args)
        {
            // escribir en la consola
            Console.WriteLine("Hola a todos !!!");
            Console.ReadKey();
        }
    }
}
```

# Intro. a la Programación con C# .NET

## Sintaxis y Semántica de C#: Variables

Una variable representa una posición en memoria para almacenar un valor.

Para usar una variable sólo hay que definirla indicando cual será su nombre y cuál será el tipo de datos que podrá almacenar, con la siguiente sintaxis:

**<tipoDeDato> <nombreVariable>;**

Veamos un ejemplo:

# Intro. a la Programación con C# .NET

## Variables

Escribir el código a continuación del anterior y ejecutar:

```
// Declaración de la variable nombre, que es de tipo string
string Nombre;
Console.Write("¿Cuál es tu nombre? ");
Nombre = Console.ReadLine();
Console.WriteLine("Me llamo {0}", Nombre);
Console.ReadKey();
```

# Intro. a la Programación con C# .NET

## Sintaxis y Semántica de C#:

### Constantes

Para las constantes el compilador también reserva un espacio de memoria para almacenar un dato, pero éste dato es siempre el mismo y no se puede modificar durante la ejecución del programa.

Un ejemplo claro sería almacenar el valor de pi en una constante para no tener que poner el número en todas las partes donde lo podamos necesitar.

Se declaran de un modo similar a las variables, con la palabra **const** en la declaración.

Veamos un ejemplo:

# Intro. a la Programación con C# .NET

## Constantes

Escribir el código a continuación del anterior y ejecutar, si lo desea PUEDE COMENTAR EL CODIGO ANTERIOR con los caracteres para bloque de comentario `/* */`

```
// Esto es una constante
const int DiasSemana=7;
const int HorasPorDia = 24;

Console.WriteLine("La cantidad de horas de una semana es {0}",
                  DiasSemana * HorasPorDia);
Console.ReadKey();
```

# Intro. a la Programación con C# .NET

## Tipos de dato

El tipo de dato indica que clase de dato tendrán las variables y las constantes, y define las operaciones que se pueden hacer con ellas.

El uso correcto de los distintos tipos de datos es fundamental para que una aplicación sea eficiente con el menor consumo posible de recursos.

Ejercitaremos con los siguientes tipos de dato:

# Intro. a la Programación con C# .NET

## Tipos de dato

Tipo de dato	Tamaño/Rango
int	Entero de 32 bit con signo desde -2,147,483,648 a 2,147,483,647
double	Número en punto flotante hasta 15-16 dígitos de precisión con rango aproximado desde $\pm 5.0 \times 10^{-324}$ a $\pm 1.7 \times 10^{308}$
float	Número en punto flotante hasta 7 dígitos de precisión con rango aproximado desde $-3.4 \times 10^{38}$ to $+3.4 \times 10^{38}$
char	1 carácter del tipo Unicode 16-bit
string	Cadenas de caracteres unicode, de longitud variable. <b>Almacena caracteres de texto y numeros como caracteres.</b>
bool	Valores lógicos true/false (booleanos)



# Intro. a la Programación con C# .NET

## Construcciones del lenguaje

Cuando se desarrolla un programa, luego de declarar las variables, constantes, clases, etc., se deben especificar instrucciones que representan el flujo de un programa.

Para C# cada línea de código termina con un “;” – punto y coma.

El tipo de instrucción básica, se denomina **sentencia**.

Normalmente, las sentencias se ponen unas debajo de otras. He aquí algunos ejemplos de sentencias:

```
int x=10;
```

```
Using System.IO;
```

```
Console.WriteLine("Hola Mundo");
```

# Intro. a la Programación con C# .NET

## Bloques de código

Un bloque de código es un grupo de sentencias que se comportan como una unidad. Se los usa para indicar el comienzo y fin de un conjunto de sentencias de código.

Por lo general, se los usa para delimitar namespaces, clases, estructuras, enumerados y métodos. Un bloque de código está limitado por las llaves de apertura “{” y cierre “}”.

He aquí un ejemplo de uso :

```
Static void main() {  
    Linea1;  
    Linea2;  
}
```

# Intro. a la Programación con C# .NET

## Ejemplos con tipo int

```
//VARIABLES ENTERAS
```

```
int a;           //defino una variable
```

```
a = 2;          //asigno valor a la variable
```

```
int b = 3;       //defino y asigno valor en una sola linea
```

```
int c = a + b;   //defino y asigno un cálculo
```

```
Console.WriteLine("Variable a tiene {0}",a);
```

```
Console.WriteLine("Variable b tiene {0}", b);
```

```
Console.WriteLine("Variable c tiene {0}", c);
```

```
Console.WriteLine("Suma es {0}", a + b + c);
```

# Intro. a la Programación con C# .NET

## Ejemplos con tipo string

```
//VARIABLES STRING
```

```
string p;
```

```
string l;
```

```
string f;
```

```
p = "Perro";
```

```
l = "ladra";
```

```
f = "fuerte";
```

```
Console.WriteLine(p + l);    //El + con string concatena
```

```
Console.WriteLine(p + " " + l);
```

```
Console.WriteLine("el {0} que {1} muy {2}", p, l, f);
```

```
Console.ReadKey();
```

# Intro. a la Programación con C# .NET

## Ejemplos con tipo char

```
// Variables Char
```

```
char char1 = 'Z';           // un character  
char char2 = (char)88;      // Valor tabla ASCII convertido a CHAR  
Console.WriteLine(char1 + " " + char2);  
Console.ReadKey();
```

# Intro. a la Programación con C# .NET

## Ejemplos con tipo bool

```
// VARIABLES Boolean

bool z;
z = true;
Console.WriteLine(z);
z = false;
Console.WriteLine(z);
Console.ReadKey();
```

# Intro. a la Programación con C# .NET

## Ejemplos con tipo float y double

```
// VARIABLES float
```

```
float fl = 0;  
Console.WriteLine(fl);  
fl = 95.60f;           //Asigna el literal 95.60 con sufijo f o F  
Console.WriteLine(fl);
```

```
// VARIABLES double
```

```
double dl = 0;  
Console.WriteLine(dl);  
dl = 95.60d;           //Asigna el literal con sufijo d o D  
Console.WriteLine(dl);
```

```
Console.ReadKey();
```

# Intro. a la Programación con C# .NET

## Operadores

Un operador en C# es un símbolo formado por uno o más caracteres que permite realizar una operación entre uno o más datos y produce un resultado.

A continuación se describen los operadores clasificados según el tipo de operaciones que permiten realizar:

Aritméticos	De asignación
Lógicos	Para cadenas
Relacionales	Condicionales



# Intro. a la Programación con C# .NET

## Operaciones aritméticas

Los operadores aritméticos incluidos en C# son los típicos de suma (+), resta (-), producto (\*), división (/) y módulo (%) También se incluyen operadores unarios: “menos unario” (−) y “más unario” (+).

```
int a = 2;
```

```
int b = 3;
```

```
Console.WriteLine("{0} más {1} es {2} ",a,b, a+b);
```

```
Console.WriteLine("{0} menos {1} es {2} ", a, b, a - b);
```

```
Console.WriteLine("{0} multiplicado por {1} es {2} ", a, b, a * b);
```

```
Console.WriteLine("{0} dividido por {1} es {2} ", b, a, b / a);
```

```
Console.WriteLine("El resto entre {0} y {1} es {2} ", b, a, b % a);
```

```
Console.ReadKey();
```

# Intro. a la Programación con C# .NET

## Operaciones lógicas

Los operadores lógicos son: “and” **&&**, “or” **||** , “not” **!** y “xor” **^**.

Analicemos la tabla de verdad para ver los resultados:

A	B	A or B	A and B	Not A	A xor B
False	False	False	False	True	False
False	True	True	False	True	True
True	False	True	False	False	True
True	True	True	True	False	False

# Intro. a la Programación con C# .NET

## Operaciones lógicas

```
//OPERACIONES LOGICAS
```

```
bool A = true;
```

```
bool B = false;
```

```
bool resultado = A && B;
```

```
Console.WriteLine("{0} and {1} da {2}", A, B, resultado);
```

```
resultado = A || B;
```

```
Console.WriteLine("{0} or {1} da {2}", A, B, resultado);
```

```
resultado = !A;
```

```
Console.WriteLine("not {0} da {1}", A, resultado);
```

```
Console.ReadKey();
```

# Intro. a la Programación con C# .NET

## Operaciones relacionales

Se han incluido los tradicionales operadores de “igualdad” `==`, “desigualdad” `!=`, “mayor que” `>`, “menor que” `<`, “mayor o igual que” `>=` y “menor o igual que” `<=`

```
//OPERADORES RELACIONALES
int A = 10;
int B = 5;
Console.WriteLine("{0} es mayor a {1} ? {2}", A, B, A>B);
Console.WriteLine("{0} es menor a {1} ? {2}", A, B, A<B);
Console.ReadKey();
```

# Intro. a la Programación con C# .NET

## Operadores de asignación/1

Para realizar asignaciones el operador **=**. Por ejemplo, la expresión **a = b** asigna a la variable **a** el valor de la variable **b**.

También existen operadores de asignación compuestos que permiten ahorrar tecleo a la hora de realizar asignaciones tan comunes como:

```
temperatura = temperatura + 15;
```

Equivale a

```
temperatura += 15;
```

# Intro. a la Programación con C# .NET

## Operadores de asignación/2

Otros dos operadores de asignación incluidos son los de incremento **++** y decremento **--**

Estos operadores permiten, aumentar y disminuir en una unidad el valor de la variable sobre el que se aplican. Así, estas líneas de código son equivalentes:

**temperatura = temperatura + 1;**

Equivale a

**temperatura += 1;**

Equivale a

**temperatura++;**

# Intro. a la Programación con C# .NET

## Operadores de asignación/3

Si el operador **++** se coloca tras el nombre de la variable devuelve el valor de la variable antes de incrementarla, mientras que si se coloca antes, devuelve el valor de ésta luego de incrementarla; y lo mismo ocurre con el operador **--**

**c = b++;** // Asigna a c el valor de b y luego incrementa b

**c = ++b;** // Incrementa b y luego se asigna a c

**Lab**: definir las variables c y b tipo entero y probar el incremento antes y después mostrando el resultado.

# Intro. a la Programación con C# .NET

## Operadores de asignación/3

### Resultado Lab

```
// Incrementos con ++
```

```
int c;
```

```
int b = 23;
```

```
c = b++; // Asigna a c el valor de b y luego incrementa b
```

```
Console.WriteLine("b++ {0}", c);
```

```
Console.ReadKey();
```

```
c = ++b; // Incrementa b y luego se asigna a c
```

```
Console.WriteLine("++b {0}", c);
```

```
Console.ReadKey();
```



# Intro. a la Programación con C# .NET

## Operaciones con cadenas

Para realizar operaciones de **concatenación** de cadenas se puede usar el mismo operador que para realizar sumas, ya que en C# se ha redefinido su significado para que cuando se aplique entre operandos que sean cadenas o que sean una cadena y un carácter lo que haga sea concatenarlos.

Por ejemplo, **"Hola"+" mundo"** devuelve **"Hola mundo"**

# Anexo

## Nombres de variables y constantes

Para nombrar variables se suele usar la notación húngara con prefijos. Estos prefijos sirven para inferir el tipo de dato en el código del programa.

Convención	Tipo de dato
str	string
int	int
dbl	double
dat	date
bln	boolean

Y también para variables el formato del nombre suele ser **CamelCase**. Las constantes se nombran en mayúsculas.

# Anexo

## Estructuras de control IF

```
// Introducción estructura IF condicional
int condicion = 3;
Console.WriteLine("Condición = " + condicion);

if (condicion == 3)
{
    Console.WriteLine("La condición se cumplio");
}
else
{
    Console.WriteLine("La condición no se cumplio");
}

Console.ReadKey();
```

Cambiar el valor 3 por otro. **Qué ocurrió?**

# Anexo

## Operadores And y Or

Para los operadores lógicos “**and**” & o && y el “**or**” | o ||

El & evalúa toda la expresión aunque ya sepa que da falso.

El && evalúa lo necesario, es más optimo en ejecución.

Usar && y || es más optimo.

Fundamental el orden de  
prioridad de las partes  
de la expresión.

Cuando comprueba que **B == 5**  
**es falso**, no sigue evaluando el  
resto de la expresión.

```
int A=30;  
int B=4;  
int Z=8;
```

Evalúa toda la  
expresión aunque  
**B==5 ya dé falso**



```
if (A>20 & B==5 & Z<100)  
{  
}
```



```
if (A > 20 && B == 5 && Z < 100)  
{  
}
```

# Anexo

## Variables y Conversiones

- Sólo se admiten conversiones entre tipos compatibles.
- No se pueden usar variables no inicializadas, la precompilación avisa.
- Sistema de tipos unificado. Todos los tipos de datos que se definan derivarán, de una clase base común llamada **System.Object**, por lo que dispondrán de todos los miembros definidos en ésta clase. Esto también es aplicable a los tipos de datos básicos.
- Compatible. C# mantiene una sintaxis muy similar a C, C++ o Java que permite incluir directamente en el código C# fragmentos de código escrito en estos lenguajes.

# Anexo

## Literales

Un literal es la representación explícita de los valores que pueden tomar los tipos básicos del lenguaje.

```
123          // Int
0x7B         // Hexadecimal
123U         // Unsigned
123ul        // Unsigned long
123L         // Long

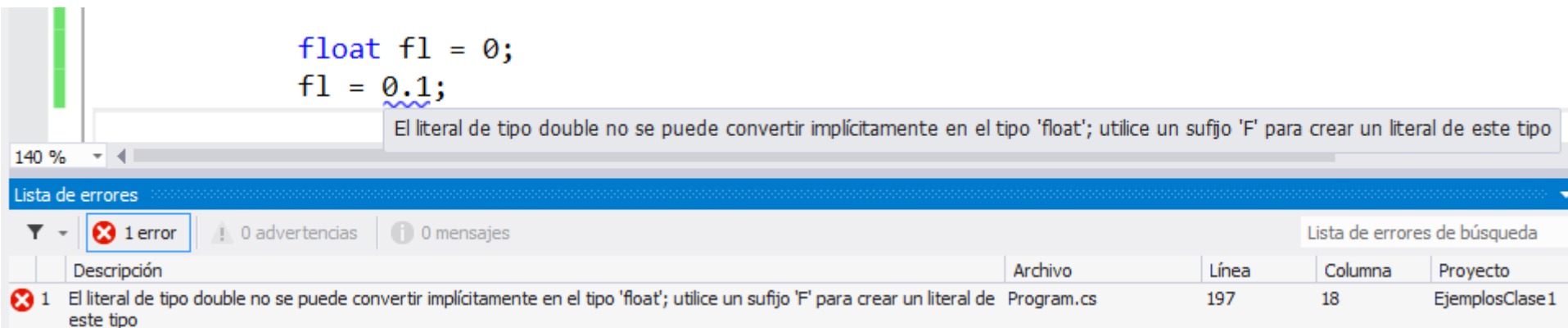
123.0        // Double
123f         // Float
123D         // Double
123.456m     // Decimal
1.23e2f      // Float
12.3E1M      // Decimal

true
false
```

# Anexo

## Variables y Literales

Todo valor real es un **double** por omisión (aunque no se indique su sufijo **d** o **D**). En la siguiente imagen al definir la variable como float, C# espera que se indique el literal del float (**f** o **F**)



```
float f1 = 0;  
f1 = 0.1f;
```

No hace falta  
el sufijo para  
el double



```
double db = 0;  
db = 0.1;
```

# Anexo

## Conversiones de tipo – objeto Convert

Implícitas	Explícitas
Ocurren automáticamente	Requieren un casting
Siempre tienen éxito	Pueden fallar
No se pierde información	Se puede perder información

```
int x = 123456;  
long y = x;           // implicit  
short z = (short)x;   // explicit  
  
double d = 1.2345678901234;  
float f = (float)d;    // explicit  
long l = (long)d;      // explicit
```

```
double dNumber = 23.15;  
int iNumber = System.Convert.ToInt32(dNumber);  
  
Console.WriteLine(iNumber);  
Console.ReadKey();
```

Convert.t

- ☞ ToBoolean
- ☞ ToByte
- ☞ ToChar
- ☞ ToDateTime
- ☞ ToDecimal
- ☞ ToDouble
- ☞ ToInt16
- ☞ ToInt32
- ☞ ToInt64

vertencias





# Anexo

## Tipos de dato y alias

```
string x="";  
x = "a" + "z";
```

```
String y = "";  
y = "a" + "z";
```

```
System.String z = "";  
z = "a" + "z";
```

Tipo System.String

Tipo String. System está mencionado en el **using System**; Se pueden usar sus tipos directamente.

Alias string del tipo String

Alias	Clase	Características
sbyte	System.Sbyte	1 byte con signo
byte	System.Byte	1 byte sin signo
short	System.Short	2 bytes con signo
ushort	System.UShort	2 bytes sin signo
int	System.Int32	4 bytes con signo
uint	System.UInt32	4 bytes sin signo
long	System.Int64	8 bytes con signo
ulong	System.ULong64	8 bytes sin signo
float	System.Single	IEEE 754, 32 bits
double	System.Double	IEEE 754, 64 bits
decimal	System.Decimal	128 bits, 96 bits valor exacto + escala
bool	System.Boolean	1 byte, 2 en arrays
char	System.Char	Unicode, 2 bytes
string	System.String	Cadenas de caracteres
object	System.Object	Cualquier objeto

# **Intro. a la Programación con C # .NET**

**Resolver los ejercicios BONUS del  
Lab 1**

**Tratar de hacerlos sólo sin mirar la  
solución!!!**

**Dudas me van consultando**