Sintaxis Básica C#

Sensitivo

- C# es case - sensitive. Es sensible a mayúsculas y minúsculas

```
// correcta
Console.WriteLine("prueba");

// incorrecta
console.writeline("prueba");


// incorrecta
Console.writeLine("prueba");
```

Terminaciones de Línea

```
// Todas las sentencias terminar con ;
string sFirstName = "";
string sLastName = "";
string sName = sFirstName + " " + sLastName;
```

Comentarios: dos tipos

```
// Comentario de una sola linea

string sFirstName = "Juan";


// Comentario de bloque

/*
string sFirstName = "";
string sLastName = "";
string sName = sFirstName + " " + sLastName;
*/
```

Variables

Variables en .NET:

- Declaradas en cualquier lugar del lugar del código.
- Todas deben tener un tipo.
- El contenido de la variable tiene que estar de acuerdo con su definición.
- Es fuertemente tipado.
- El tipo de variable precede al indicador
- Deben ser indicializadas antes de ser usadas.

Declaración de Variables

```
// Definicion de variables
string sFirstName = "Juan";
int contador = 0;
bool exito = true;
decimal importe = 0;
DateTime nacimiento;
Cliente cliente;
```

Tipos Integrados Simples

Tipo C#	Intervalo	Tamaño / Precisión	Tipo .NET	Default
`sbyte`	De -128 a 127	Entero de 8 bits con signo	`System.SByte`	0
`byte`	De 0 a 255	Entero de 8 bits sin signo	`System.Byte`	0
`short`	De -32 768 a 32 767	Entero de 16 bits con signo	`System.Int16`	0
`ushort`	De 0 a 65.535	Entero de 16 bits sin signo	`System.UInt16`	0
`int`	De -2.147.483.648 a 2.147.483.647	Entero de 32 bits con signo	`System.Int32`	0
`uint`	De 0 a 4.294.967.295	Entero de 32 bits sin signo	`System.UInt32`	0
`long`	De -9.223.372.036.854.775.808 a 9.223.372.036.854.775.807	Entero de 64 bits con signo	`System.Int64`	0

Tipos Integrados Simples

`ulong`	De 0 a 18.446.744.073.709.551.615	Entero de 64 bits sin signo	`System.UInt64`	0
`float`	De ±1,5 x 10^{-45} a ±3,4 x 10^{38}	7 dígitos	`System.Single`	0.0f
`double`	De ±5,0 × 10^{-324} a ±1,7 × 10^{308}	15-16 dígitos	`System.Double`	0.0d
`decimal`	De ±1,0 x 10^{-28} to ±7,9228 x 10^{28}	28-29 dígitos significativos	`System.Decimal`	0m
`char`	U+0000 a U+FFFF	Carácter Unicode de 16 bits	`System.Char`	\x0000
`bool`	Booleano	true, false	`System.Boolean`	false

Tipos Integrados No Simples

Tipo C#	Descripción	Tipo .NET
`object`	Es la clase base para todos los demás tipos, incluidos los tipos integrados simples.	`System.Object`
`string`	Una secuencia de caracteres Unicode.	`System.String`
`dynamic`	Es un tipo diseñado para ser usado con assemblies escritos en lenguajes dinámicos	No corresponde a un tipo .NET

Conversión de tipos

Cuando convertimos de un tipo superior a otro inferior hay riesgo de perdida de información.

Por ejemplo un tipo «decimal» a otro tipo «int«.
Para limitar este riesgo el compilador activa un error cuando se encuentra con esta situación.

```
// Nos da error
 int contador = 0;
 decimal importe = (decimal)1000.80;


 contador = importe;
```

Conversión de tipos

Es necesario hacer una conversión explícita de tipos para indicarle al compilador que efectivamente queremos realizar la conversión.

Conversión explícita:

Casting: Consiste en indicar entre paréntesis el tipo de dato al cual queremos convertir el valor

Si falla el casting se genera una InvalidCastException

```
// Nos da error
 int contador = 0;
 decimal importe = (decimal)1000.80;


 // casteo a int
 contador = (int)importe;
```

Conversión de tipos

- **Para solicitar un número al usuario utilizo ReadLine, este método devuelve un string.**

- Debo parsear el string a un número. Que pasa si pongo letras?

```
// Solicito un número, pero debe ser correcto
int numero;
numero = int.Parse(Console.ReadLine());
```

La mejor opción es usar TryParse para evitar que de error.

```
// Solicito un número, si hay error numero queda en 0
int numero;
int.TryParse(Console.ReadLine(), out numero);
```

Ingresar datos

- **Para solicitar un número al usuario utilizo ReadLine, este método devuelve un string.**

- Debo convertir el string a un número.
- int numero;
- numero = int.Parse(Console.ReadLine());

Que pasa si pongo una letra? Me da error, puedo utilizar el tryparse

bool exito = int.TryParse(Console.ReadLine(), out numero);

Si es un número, el mismo va a quedar cargado en la variable númeroy exito vale true.

Si es una letra, número va a quedar en 0 y exito vale false.

Excepciones

La sentencia para manejo de errores Try-Catch en C# es utilizado en programación .Net para evitar romper el flujo de trabajo de una aplicación.

La instrucción Try-Catch esta formado de un bloque try y seguida de este se coloca el catch.

Try: se encarga de encapsular todas las operaciones.

```
try
{
   //boque try con todas las operaciones
}
```

Catch: captura los errores generados en el bloque Try, aqui se manejan las diferentes excepciones.

```
catch (Exception ex) //bloque catch para captura de error
{
   //acción para manejar el error
}
```

Conversión de tipos

```
int numero;
try
{
    Console.WriteLine("Ingrese un número");
    numero = int.Parse(Console.ReadLine());
    Console.WriteLine("El numero ingresado es" + numero);
}
catch (Exception)
{
    Console.WriteLine("EL número ingresado no es válido.");
}
```

Estructuras de Decisión

```
int numero = 5;
if (numero == 0)
{
    Console.WriteLine("El número es 0");
}
else if (numero < 0)
{
    Console.WriteLine("El número es menor a 0");
}
else
{
    Console.WriteLine("El número es " + numero);
}
```

Estructuras de Decisión

```
string pais = "Brasil";
string deporte = "";

switch (pais)
{
    case "Brasil":
        deporte = "Fultbol";
        break;

    case "USA":
        deporte = "Basquet";
        break;

    default:
        deporte = "Tenis";
        break;
}
Console.WriteLine("El el pais " + pais + " el deporte principal es " + deporte);
```

Operadores Lógicos

C#	Operador
&&	**Operador logico Y**
||	**Operador logico O**
!	**Negacion logica**
==	**Igual**
!=	**Distinto**

```
//Si Hacer1() es True, entonces
//NO se evalua Hacer2()
if (Hacer1() || Hacer2())
{
}
```

```
//Si Hacer1() es False, entonces
//NO se evalua Hacer2()
  if (Hacer1() && Hacer2())
  {
  }
```

Estructuras de iteración

```
// muestra de 0 a 9
for (int i = 0; i < 10; i++)
{
    Console.WriteLine("El numero es : " + i );
}
```

Estructuras de iteración

```
// con un array de int
int[] fibNumbers = { 0, 1, 2, 3, 5, 8, 13 };
foreach (int numero in fibNumbers)
{
    if (numero % 2 == 0)
    {
        Console.WriteLine("El número " + numero + " es par");
    }
    else
    {
        Console.WriteLine("El número " + numero + " es impar");
    }
}
```

Estructuras de iteración

```
int i = 0;
bool check = true;

while (check)
{
    Console.WriteLine("El numero es : " + i);
    i++;
    if (i >= 10)
    {
        check = false;
    }


}
```

```
int i = 0;
bool check = true;

do
{
    Console.WriteLine("El numero es : " + i);
    i++;
    if (i >= 10)
    {
        check = false;
    }
} while (check);
```

Concatenar string

Muchas veces temenos que mostrar textos con palabras y el contenido de Variables.

String Interpolation

Se pueden incluir el nombre de una variables o cadena de caracteres. De esta manera, en tiempo de ejecución, este nombre de la variable será sustituido por su correspondiente valor..

```
string nombre = "Juan";
int edad = 18;

// concatenacion de string
Console.WriteLine("Hola mi nombre es " + nombre + " y tengo " + edad + " años");

// interpolacion de string
Console.WriteLine($"Hola mi nombre es {nombre} y tengo {edad} años");
```

Operador ternario

- Tiene 3 partes:

- La expresión booleana; La cual nos devolverá verdadero o falso.

- Sentencia 1; es la expresión que se va a devolver en caso de que la expresión booleana sea true. (antes de los dos puntos :)

- Sentencia 2; Es la expresión que se va a devolver en caso de que la expresión devuelve false.

```
Expresión booleana ? sentencia 1 : sentencia 2;
```

Operador nullable

- Mediante el ?? puede saber si su valor es null, quedando mas limpio el código

```
0 references
public Auto DarAuto(string matricula)
{
    // usando el operador nullable, puedo devolver el auto encontrado o un nuevo auto
    //
    Auto unAuto = Auto.BuscarAuto(matricula);
    return unAuto ?? new Auto();
}
```

```
// usando el operador nullable, puedo mostrar el dato correcto

string nombre = null;
Console.WriteLine(nombre ?? "No tiene nombre");
```

Arrays

- C# utiliza corchetes [ ] para definición de arrays

```
//Definicion de un Arreglo de strings
string[] telefonos;

//De 3 elementos
telefonos = new string[3];

//Seteo del 1er elemento del arreglo
telefonos[0] = "1245";

//Definicion y asignacion de una vez
telefonos = new string[] {"1","2","3"};
```