

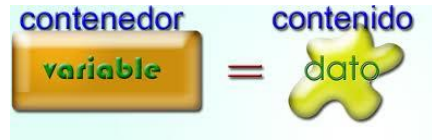
Tema 3: Variables

Aplicaciones Ofimáticas. CM Sistemas Microinformáticos y Redes



Concepto de variables.

- Los programas necesitan datos para poder realizar los cálculos que les pedimos.
- Estos datos deben ser almacenados para que puedan ser accedidos por el programa cuando los necesite.
- Las variables estarán formadas por un espacio en la memoria y un nombre para referirse a dicho espacio:



- La sintaxis para **declarar una variable** en C# es la siguiente:

`[Tipo_Variable] [nombre_variable] ; → int x;`

- Tenemos una variable de nombre **x** y de tipo Integer.

`double` dob = 19.6 → Mal nombre

`double` temperatura = 19.6 → Bien

- Ojo: **Los nombres de las variables deben ser representativos del valor que estamos guardando.**

Concepto de variables

- Existen diferentes tipos de datos que ocuparán más o menos memoria.
- En general pueden clasificarse en :
 - **numéricos:**
 - Enteros
 - Reales
 - **alfanuméricos**
 - letras
 - caracteres especiales
 - números: dni , número de teléfono ...
 - Cualquier mezcla de los anteriores
 - **booleanos:** Verdadero o falso.

Tipos de datos en C#

- En función del dato que queramos almacenar en memoria utilizaremos un tipo de variables u otras.

Tipo C#	Espacio	Valores	Valor por defecto
Byte	1 Byte	0 a 255	0
SByte	1 Byte	-128 a 127	0
Short	2 Bytes	de -32,768 a 32,767.	0
Integer	4 Bytes	de -2,147,483,648 a 2,147,483,647	0
Long	8 Bytes		0
Single	4 Bytes	Decimales precisión simple	0
Double	8 Bytes	Decimales precisión doble	0
Decimal	16 Bytes	Más capacidad, más precisión	0
Char	2 Bytes	Un caracter	" "c
String	-----	Cadena de caracteres	Nothing
Boolean	2 Bytes	Valor verdadero o falso	False

Declaración e inicialización de variables.

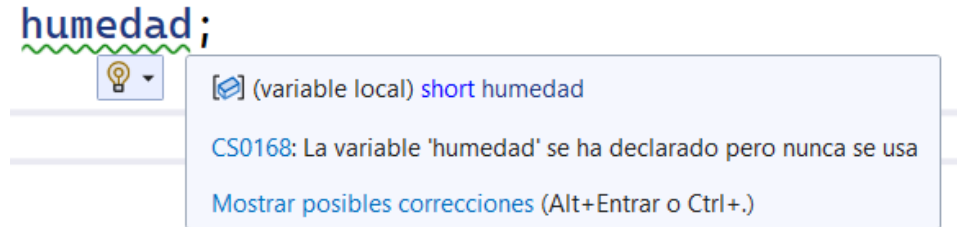
- Las variables deben de ser declaradas al principio del programa, evitando que aparezcan dispersas por el código.

```
int x, y, z;  
short temp, regist, humedad;
```

- Cuando queramos declarar varias variables del mismo tipo podemos hacerlo separando sus nombres por comas.
- Cuando declaramos una variable, esta almacena un valor por defecto.

Declaración e inicialización de variables.

- Si no le damos otro valor en el programa, nos avisa de que no estamos utilizando esa variable, subrayándola en verde. Esto es simplemente una advertencia, no un error.



- En caso de que queramos asignarle otro valor utilizaremos el operador =

`//Inicializo la variable`

`double dob = 19.6;`

`//Modifico el valor de la variable`

`dob = 50.6;`

Nombres de variables:

- Los nombres de las variables deben cumplir los siguientes requisitos:

- No deben contener espacios.

Ej: sueldo neto (Mal) → sueldoNeto (Bien)

- No pueden comenzar por números , aunque si pueden contener números.

Ej: 1x (Mal) → x1 (Bien)

- No pueden contener caracteres como *, \$, &, . El guión bajo si está aceptado.

Ej: &x (Mal) → _x (Bien)

- Cuando el nombre de una variable está formado por dos palabras. la segunda comienza en mayúsculas.

Ej: presionVapor , sueldoNeto, valorAbsoluto, raizNumero

Ejemplo 2.1: Suma de tres variables

- Construiremos un programa que almacene 3 números en tres variables de tipo short y calcule la suma guardándola en una variable de tipo Integer.
- A continuación el programa informará al usuario del valor de la suma:

```
//Declaramos las variables que vayamos a utilizar
//Tres variables de tipo short para los números

short num1, num2, num3;


//Inicializamos las variables a los valores que queremos sumar
num1 = 25;
num2 = 30;
num3 = 35;

//Una variable de tipo Integer para guardar el resultado, la suma
int suma;

//Indicamos la suma de los 3 números
suma = num1 + num2 + num3;

//Mostramos por pantalla la suma
Console.WriteLine("La suma es : " + suma);
Console.ReadLine();
```

*Para mezclar texto con variables
debemos de utilizar el operador +*



Constantes.

- Cuando queremos guardar un valor que sabemos que nunca va a ser modificado, podremos guardarlo en una constante.
- La sintaxis es similar a la vista para las variables:

```
const double pi = 3.14151623;
```

- Una constante ya nunca podrá ser modificada ya que dará error de compilación:

```
const double pi = 3.14151623;
```

```
pi = 9;
```

 (constante local) double pi = 3.14151623

CS0131: La parte izquierda de una asignación debe ser una variable, una propiedad o un indizador

[Mostrar posibles correcciones \(Alt+Entrar o Ctrl+.\)](#)

Lecturas de datos por teclado.

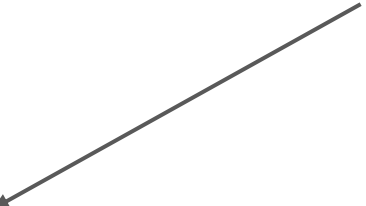
- Para solicitar al usuario que introduzca los datos por teclado debemos realizar 2 pasos:
 - 1º: Enviaremos un mensaje informándole del dato que queremos que introduzca
 - 2º: Guardaremos el dato en la variable mediante la función `Console.ReadLine`

Lecturas de datos por teclado.

- Para solicitar al usuario que introduzca los datos por teclado debemos realizar 2 pasos:

```
string nombre;  
int edad;  
  
//Solicitamos al usuario el dato  
Console.WriteLine("Introduzca el nombre");  
  
//Leemos el dato con ReadLine  
nombre = Console.ReadLine();  
  
//Solicitamos la edad  
Console.WriteLine("Introduzca su edad: ");  
edad = Convert.ToInt32(Console.ReadLine());  
  
//Mostramos los datos al usuario, simplemente para verificar  
Console.WriteLine("Bienvenido: " + nombre + " tiene usted " + edad + " años");  
Console.ReadLine();
```

La función `Console.ReadLine` siempre lee el dato como si fuera un **String**, por eso cuando queramos leer otro tipo de variables, como en este caso la edad, deberemos convertirla en el tipo adecuado.



Ejemplo 2.2: Lectura datos teclado

- Solicitaremos todos los datos de un determinado cliente: DNI, Nombre, edad y saldo:

```
//Declaramos variables que necesitamos
string dni, nombre;
short edad;
int saldo;

//Solicitamos los datos
Console.WriteLine("Introduzca el DNI:");

//Guardamos el dato que va a introducir en la variable DNI
dni = Console.ReadLine();
Console.WriteLine("Introduzca el nombre: ");
nombre = Console.ReadLine();
Console.WriteLine("Introduzca la edad: ");

//Guardamos la edad convirtiéndola previamente en una variable de tipo Short
edad = Convert.ToInt16(Console.ReadLine());
Console.WriteLine("Introduzca el saldo: ");
saldo = Convert.ToInt32(Console.ReadLine());
```

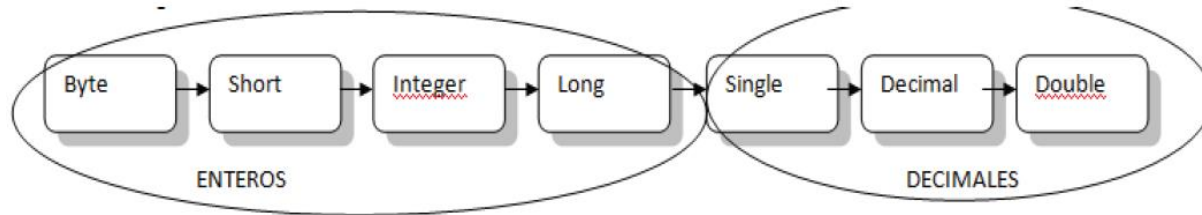
Ejemplo 2.2: Lectura datos teclado

- Mostraremos esos datos por pantalla:

```
Console.WriteLine("Sus datos son los siguientes: ");  
Console.WriteLine("DNI: " + dni);  
Console.WriteLine("Nombre: " + nombre);  
Console.WriteLine("Edad: " + edad);  
Console.WriteLine("Saldo: " + saldo);  
Console.ReadLine();
```

Conversión de tipos.

- Cada variable ocupa un espacio en memoria en función del tipo de dato que queramos guardar en ella.
- Esto les proporciona mayor o menor precisión según sea el caso.
- Podemos ordenar las variables de menor a mayor precisión de la siguiente manera:



Conversión de tipos.

- Las variables de menor precisión ***siempre podrán ser guardadas*** en las variables de mayor precisión(salvo la excepción de Decimal)

```
byte numByte = 220;  
short numShort = numByte;  
int numInt = numShort;  
long numLong = numInt;
```

```
float numSingle = 10.5f;  
double numDouble = numSingle;  
decimal numDecimal = numDouble; //Da error
```

- Este tipo de conversiones se conocen como conversiones implícitas porque es el lenguaje el que se encarga de hacerlas automáticamente,

Conversión de tipos

- Las conversiones entre tipos primitivos se realizan **cuando en una expresión aparecen operandos de diferente tipo o bien asignamos a una variable un valor de distinto tipo al definido para la variable.**
- Por ejemplo, para poder sumar dos variables hace falta que ambas sean del mismo tipo. Si una es int y otra float, el valor de la primera se convierte a float antes de realizar la operación.

Conversión de tipos

- La sintaxis es:

```
(tipo) expresión
```

- Una vez evaluada la expresión, se convierte dicho valor resultante al tipo que hayamos señalado entre paréntesis.

Conversión implícita vs Conversión explícita

Conversión implícita

- Ocurre de forma automática.
- Siempre tiene éxito.
- Nunca se pierde información en la conversión.

Conversión explícita

- Requiere usar **cast** o **Convert.Toxxx(var)**
- Puede no tener éxito.
- Puede perderse información en la conversión.



Conversión implícita vs Conversión explícita

```
int intValor = 123;  
long longValor = intValor; // implícita de int a long  
intvalor = (int) longValor; // explícita de long a int con cast  
int x = 123456;  
long y = x; // implícita  
short z = (short)x; // explícita  
double d = 1.2345678901234;  
float f = (float)d; // explícita  
long l = (long)d; // explícita  
//uso de la clase Convert:  
int c=154;  
string s = Convert.ToString(c);
```

Conversión implícita segura

- La conversión implícita es segura entre estos tipos:

Tipo	Se convierte de forma segura a:
Byte	short, ushort, int, uint, long, ulong, float, double, decimal
Sbyte	short, int, long, float, double, decimal
Short	int, long, float, double, decimal
Ushort	int, uint, long, ulong, float, double, decimal
Int	long, float, double, decimal
UInt	long, ulong, float, double, decimal
Long	float, double, decimal
Ulong	float, double, decimal
Float	double
Char	ushort, int, uint, long, ulong, float, double, decimal

Conversión utilizando Parsing

- Permite poder convertir la información contenida en un string a un tipo de dato.
- Esta acción es facilitada por el método Parse() que implementan los tipos numéricos y de fecha.

```
string numero= "1234";  
int i= int.Parse(numero);  
long l = long.Parse(numero);  
byte b = byte.Parse(numero); // OverflowException
```

Conversión utilizando Parsing

- C# dispone de un método para realizar la conversión realizando una comprobación utilizando un método booleano **.TryParse()**.
- Es el **MÁS RECOMENDADO**, pues controla las excepciones

```
double numero;  
string texto = "25.542";  
  
if (Double.TryParse(texto, out numero))  
{  
    Console.WriteLine("'{0}' --> {1}", texto, numero);  
}  
else  
{  
    Console.WriteLine("No se puede parsear '{0}'.", texto);  
}
```

Funciones librería Math

- La librería **System.Math** incluye una serie de funciones que nos serán muy útiles para simplificar los cálculos matemáticos:
 - **Math.sqrt**(numero) as **Double**: Calcula la raíz del número que se le pasa como parámetro y la devuelve como un Double.
 - **Math.Pow**(num1,num2) as Double: Calcula el primer numero elevado al segundo que se le pasan como parámetro. El dato que devuelve es de tipo Double.

Funciones librería Math

- **Math.round(num): Redondea al entero más cercano**
- **Math.Round**(num, digitos) as Double: Redondea el número decimal que recibe como parámetro
- **Math.Floor**(num) as Double: Redondea al número entero más bajo.
- **Math.Ceiling**(num) as Double: Redondea al número entero más alto.
- **Math.Min**(num1,num2)
- **Math.Max**(num1,num2)

Funciones librería Math

```
double raiz = Math.Sqrt(100);  
double potencia = Math.Pow(10, 2);  
double redondeo = Math.Round(3.54, 1);  
double suelo = Math.Floor(4.67);  
double techo = Math.Ceiling(4.67);  
double minimo = Math.Min(suelo, techo);  
double maximo = Math.Max(suelo, techo);
```

Actividad 1

Crea un programa donde ingreses 5 números por pantalla y muestres su promedio.

¿Debemos usar conversión implícita o explícita?

Controla qué pasaría si introducimos una letra.

Tema 3: Variables

Aplicaciones Ofimáticas. CM Sistemas Microinformáticos y Redes

