

A decorative graphic on the left side of the slide, consisting of white lines and circles on a blue gradient background, resembling a circuit board or data flow diagram.

TIPIFICACIÓN DE DATOS EN C#

COMPARACIÓN EN C

- En C tenemos los tipos de datos primitivos. En C# vamos a encontrar clases que se comportan como tales.
- Acá muchas de las variables ya vienen inicializadas en 0.

The background is a gradient of blue, transitioning from a lighter shade at the top to a darker shade at the bottom. White, stylized circuit-like lines and circles are scattered along the left and right edges, resembling a network or data flow diagram.

ENTEROS

TIPOS DE DATOS ENTEROS

- Al igual que en C, debemos elegir que tipo de dato usar para minimizar el espacio de memoria del programa.
- La precisión es del 100% en todos.
- Las variables o constantes numéricas se inicializan en 0 automáticamente.

TIPOS DE DATOS ENTEROS

Descripción	Nombre C#	Nombre .Net	Precisión	Valor Mínimo	Valor Máximo	Espacio de Almacenamiento
Entero Octeto	sbyte	System.SByte	100%	-128	127	1 Byte (8 bits)
Entero Octeto Sin Signo	byte	System.Byte	100%	0	255	1 Byte (8 bits)
Entero Corto	short	System.Int16	100%	-32.768	32.767	2 Bytes (16 bits)
Entero Corto Sin Signo	ushort	System.UInt16	100%	0	65.535	2 Bytes (16 bits)
Entero (por defecto)	int	System.Int32	100%	-2.147.483.648	2.147.483.647	4 Bytes (32 bits)
Entero Sin Signo	uint	System.UInt32	100%	0	4.294.967.295	4 Bytes (32 bits)
Entero Largo	long	System.Int64	100%	-9.223.372.036.854.775.808	9.223.372.036.854.775.807	8 Bytes (64 bits)
Entero Largo Sin Signo	ulong	System.UInt64	100%	0	18.446.744.073.709.551.615	8 Bytes (64 bits)

USO EN C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

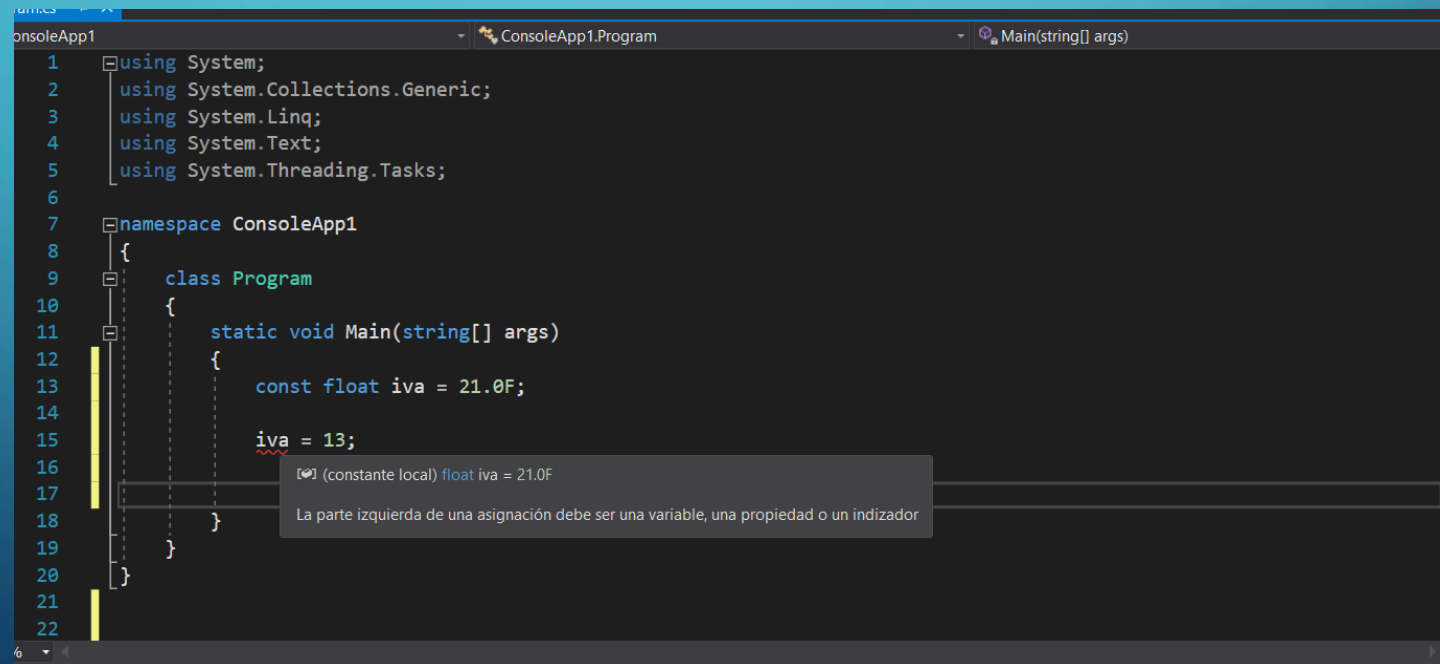
namespace ConsoleApp1
{
    class Program
    {
        static void Main(string[] args)
        {
            System.Byte var1;
            byte var2;
            sbyte var3;
            var1 = 2;
        }
    }
}
```

CONSTANTES NUMÉRICAS

- Cualquier constante numérica entera el compilador la asumirá como int.
 - `var = 2;`
- Para que sea uint debemos asignarle el identificador U.
 - `var = 2U;`
- Para que sea uint debemos asignarle el identificador L.
 - `var = 2L;`
- Para que sea ulong debemos asignarle el identificador UL
 - `var = 2UL;`

EL CONST

- Al igual que en C podemos definir constantes.
- A las constantes se les asigna un valor fijo que no puede ser cambiado en tiempo de ejecución.



```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace ConsoleApp1
8 {
9     class Program
10     {
11         static void Main(string[] args)
12         {
13             const float iva = 21.0F;
14
15             iva = 13;
16
17         }
18     }
19 }
20
21
22
```

(constante local) float iva = 21.0F

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

A decorative graphic on the left side of the slide, consisting of a network of white lines and small circles on a blue gradient background, resembling a circuit board or a neural network.

NUMÉRICOS REALES

TIPOS DE DATOS NUMÉRICOS REALES

- La selección acá debe darse por la precisión deseada para el programa. Siempre teniendo en cuenta la minimización del espacio de almacenamiento.
- Puede existir un margen de error entre el valor ingresado y el almacenado.
- Se inicializan por defecto automáticamente en 0.0

TIPOS DE DATOS NUMÉRICOS REALES

Descripción	Nombre C#	Nombre .Net	Precisión	Rango de Valores Admitidos	Espacio de Almacenamiento
Punto Flotante de Precisión Simple	float	System.Single	La más baja. En el orden del 7° dígito decimal de relevancia.	Aprox. desde $-3,4 \times 10^{+38}$ hasta $-1,4 \times 10^{-45}$, el 0 y desde $1,4 \times 10^{-45}$ hasta $3,4 \times 10^{+38}$.	4 Bytes (32 bits)
Punto Flotante de Precisión Doble	double	System.Double	Intermedia. En el orden del 15° o 16° dígito decimal de relevancia.	Aprox. desde $-1,8 \times 10^{+308}$ hasta $-4,9 \times 10^{-324}$, el 0 y desde $4,9 \times 10^{-324}$ hasta $1,8 \times 10^{+308}$.	8 Bytes (64 bits)
Punto Flotante de Precisión Extendida	decimal	System.Decimal	La más alta. En el orden del 28° o 29° dígito decimal de relevancia.	Aprox. desde $-7,9 \times 10^{+28}$ hasta $-7,9 \times 10^{+28}$ (menor rango y altísima precisión). Uso en cálculos de financieros y de precisión científica.	16 Bytes (128 bits)

INTERPRETACIÓN DE CONSTANTES

- Las constantes en números reales por defecto serán Double. También les podemos asignar el identificador D.
- Las constantes con el identificador F serán float.
- Las constantes con el identificador M serán decimal.

CARACTERES

- Acá queremos asignar caracteres individuales.
- Usa el standard UNICODE (no ASCII). Tiene 65536 caracteres en lugar de los 256 de ASCII. A por ejemplo es el mismo valor que para ASCII (65).
- Se inicializan por defecto con el fin de cadena (`\0`).
- Las constantes se explicitan usando comillas simples o apóstrofes.
 - `char variable = 'A';`

CARACTERES

Descripción	Nombre C#	Nombre .Net	Empleo	Valores Admitidos	Espacio de Almacenamiento
Caracter	char	System.Char	Almacenamiento y representación de caracteres.	Entero corto sin signo (16 bits) representando el código de asignado a al carácter el estándar UNICODE.	2 Bytes (16 bits)

CADENAS DE CARACTERES

- Acá no debemos definir los vectores de caracteres si no que tenemos el tipo string.
- String se define como una colección dinámica por lo que irá adaptando su tamaño.
- Las variables string NO TIENEN inicializador por defecto. Es null.

CADENAS DE CARACTERES

Descripción	Nombre C#	Nombre .Net	Empleo	Valores Admitidos	Espacio de Almacenamiento
Cadena de Caracteres	string	System.String	Almacenamiento y representación de cadenas de caracteres.	Cadenas de caracteres de hasta aproximadamente 2.000.000.000 caracteres UNICODE de extensión.	20 Bytes (160 bits) más dos Bytes (16 bits) por caracter.

INTERPRETACIÓN DE CONSTANTES

- Las cadenas de caracteres se expresan con comillas
 - `string variable = "hola";`
- En caso de que necesitemos usar algún carácter especial o secuencia de escape (los `\n`, `\t` y esos) debemos usar el `@` para aclarar que es una cadena textual
 - `string variable = @"nhola";`

DATOS LÓGICOS

Descripción	Nombre C#	Nombre .Net	Empleo	Valores Admitidos	Espacio de Almacenamiento
Lógico o booleanos	bool	System.Boolean	Almacenamiento y representación de los datos lógicos verdadero (true) o falso (false).	true o false	1 Byte (8 bits).

- Almacenan los valores lógicos verdadero o falso.
- Automáticamente se inicializan como false. ¿Si se declaran como bool? Se inicializan en null.

OBJETOS

- Podemos declarar variables del tipo object.
- Object puede luego ser cualquier otra clase o elemento que necesitamos.
- Si creo una variable que aún no se que puede contener podemos hacerlo con una variable del tipo Object.

```
namespace ConsoleApp1
{
    class Program
    {
        static void Main(string[] args)
        {
            Object comodin;
            comodin = 21F;
            comodin = "HOLA MAMA";
        }
    }
}
```

ENUMS

- Es un tipo de valor definido por un conjunto de constantes con nombre y un número.
- Sirve para asignar un nombre a constantes y validarlas rápidamente. Por ejemplo, si quiero definir una variable con los días de la semana o con los meses del año.

DEFINIR UN ENUM

```
namespace ConsoleApp1
{
    enum estado
    {
        pendienteDePago,
        pagado,
        enviado,
        recibido,
        error=999
    }
}
```

```
static void Main(string[] args)
{
    estado estadoOrden = estado.enviado;
    Console.WriteLine(estadoOrden);
    Console.ReadLine();
}
```

CONVERSIONES DE TIPO

CASTEOS

- Para castear una variable es similar a como se realiza en C. Podemos usar la siguiente secuencia:

```
int entero;
```

```
float flotante;
```

```
entero = 2;
```

```
flotante = (float)entero;
```

```
flotante = System.Convert.ToSingle(entero);
```

CONVERTIR A STRING

```
string texto;  
int numero = 34;  
texto = Convert.ToString(numero);  
texto = numero.ToString();  
Console.WriteLine("Quiero mostrar el numero: " + numero.ToString());
```


DE STRING A DATOS NUMÉRICOS: LA FUNCIÓN PARSE

```
int datoNumerico;  
string datoTexto = "35";  
datoNumerico = int.Parse("23");  
datoNumerico = int.Parse(datoTexto);  
datoNumerico = int.Parse(Console.ReadLine());
```

LA FUNCIÓN TRYPARSE

- La diferencia con parse es que esta devuelve un booleano para indicar si la conversión fue exitosa o no.
- Nos permite manejar excepciones o errores en tiempo de ejecución

```
bool resultado;  
int datoEntero;  
string datoString = "rompe todo";  
  
resultado = int.TryParse(datoString, out datoEntero);  
  
if (resultado == false)  
{  
    Console.WriteLine("No se pudo asignar, se rompio todo");  
}
```

OPERACIONES DE TIPOS

PRIORIDADES

- Si realizamos una operación así: $\text{tipoA} * \text{tipoB}$ el resultado depende del tipo de dato más “exigente” en cuanto a precisión.
 - $\text{var} = 2 + 3.0F$ //el resultado será float
 - $\text{var} = 2 * 3U$ //el resultado será int