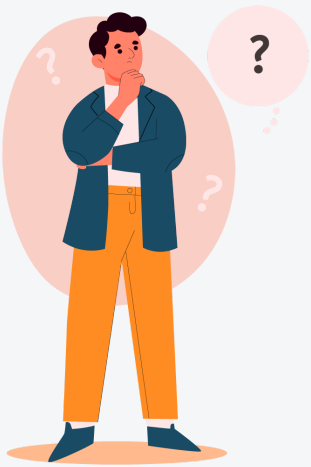
Variables y tipos de datos. Entrada y salida

Sitio: Agencia de Habilidades para el Futuro
Curso: Desarrollo de Sistemas Orientado a Objetos 1º D
Libro: Variables y tipos de datos. Entrada y salida

Imprimido por: Eduardo Moreno
Día: miércoles, 19 de marzo de 2025, 23:05

Tabla de contenidos

Preguntas orientadoras



- ¿Qué es la entrada de datos en un programa y cómo se obtiene en C#?
- ¿Cuáles son los métodos más comunes para leer datos de entrada del usuario en C#?
- ¿Cómo se muestra información o resultados en la salida de un programa en C#?
- ¿Cuál es la diferencia entre imprimir información en la consola y mostrar datos en una ventana de aplicación?
- ¿Qué técnicas se pueden utilizar para formatear la salida de datos y hacerla más legible y presentable?

¿Qué son las variables?

Son elementos que se emplean para almacenar y hacer referencia a valores. Gracias a estas es posible crear "programas genéricos" que funcionan siempre igual, independientemente de los valores concretos utilizados. Técnicamente, una variable es un espacio reservado en memoria con nombre y tipo para guardar un valor que puede cambiar

	Analicemos un ejemplo

Si quisiéramos hacer 3+1 y guardar el resultado, lo que haríamos es simple:

resultado = 3 + 1 ..... ¿no es cierto?

Ahora bien, ¿qué problema tiene nuestra solución?

- ¿Cómo resolverían este problema si quisiéramos hacer que en resultado se guarde la suma entre dos números cualesquiera?

Podríamos utilizar variables para almacenar y referirse a cada número:

numero1 = 3

numero2 = 1

resultado = numero1 + numero2

En este ejemplo entonces, los elementos numero1 y numero2 son variables que almacenan los valores que utiliza el programa. Si se modifica el valor de estas, el programa sigue funcionando correctamente. Es decir, en la variable resultado tendríamos el valor de la suma entre dos números cualesquiera, tal como queríamos.

¿Cómo crear una variable?

Una variable tiene un nombre , un tipo y guarda un valor.

Para declararlas (crearlas) debemos respetar un orden específico:

tipo nombreVariable = valor

int edadPersona = 5

• Se declaran una única vez . Luego, simplemente se utilizan o se le modifica su valor

• Se declaran (crean) dentro del *main*() (por ahora!)

Características y funcionalidades de las variables

1. Sintaxis de las variables

Puede usarse el español para nombrarlas, pero el nombre NO puede:

- Empezar con un número.
- Contener espacios.
- Contener acentos.
- Contener ñ.
- Contener caracteres especiales.
- Utilizar palabras reservadas del lenguaje
- Tener en cuenta que son sensibles a mayúsculas/minúsculas.

2. ¿Qué tipos de variables hay?

Tipo de dato	Qué guardo	Ejemplo
int	entero	int edad = 25
double	decimal	double peso = 2.3
char	caracter	char letra = 'a'
string	cadena de caracteres	string nombre = "Emma"
bool	verdadero/falso	*bool* abierto = false

3. ¿Qué podemos hacer?

Acción	Ejemplo	Observación
Declararlas y asignarles un valor	int edad = 5	

Asignar otro valor	edad=10	En este caso no ponemos int delante de edad ya que fue declarada anteriormente
Ingresar un valor por teclado y asignárselo	edad = *int.Parse (Console.ReadLine())*	Se explica mas adelante esta línea
Edad	= edad+5	Incrementa en 5 su valor
Mostrar su valor por pantalla	*Console.WriteLine*	El valor de la variable edad es:" + edad

Tipos de datos en CSharp (C#)

Conociendo el entorno

Cuando trabajamos en lenguaje c# (CSharp), cada dato es de un tipo concreto. Este tipo de dato define el conjunto de valores válidos que el dato puede tomar y el conjunto de operaciones que se pueden realizar con él.

Si vamos a almacenar una edad de un humano (0 a 120 años), pensando en el tamaño del almacenamiento: ¿en qué tipo de dato numérico lo almacenaremos *(byte, short, int o long,* que explicaremos a continuación)?

Importante:

Hay que tener en cuenta que cada tipo de datos ocupa un espacio en la memoria , y si no tenemos en cuenta ese espacio (para crecimiento a futuro, o porque estamos desperdiciando espacio) seguramente las aplicaciones tiendan a ser más lentas o el espacio de almacenamiento, tanto en memoria como en el las unidades, se llenen con espacio que no se pueda utilizar innecesariamente. Hay que elegir el espacio acorde a su rango inferior y superior y a su futuro crecimiento. En este caso un byte (de 0 a 255 valores diferentes).

En los siguientes sub capítulos veremos en detalle las diferentes categorías de tipos de datos y espacio que ocupan.

Tabla de tipos de datos en C#

Tipos de datos	Nombre	C#

Primitivos	Numéricos	int a = 10;
		int b = 7;
		int c=a+b;
		int c=a-b;

	Lógicos (Booleanos)	bool r =a > b;
		bool a=false;
		bool b=true;
		bool c=a && b;
		bool d=a || b;
		bool e= !a;

	Caracteres	char a='a'; char b='d';

Compuestos y agregados	Array	int[] a=new int[5];
		a[0] = 10;
		a[1] = -2;
		a[2] = 5;

	Cadena	string a="Hola Mundo!"; a.Length; bool r=String.Compare(s1 ,s2);
	Estructuras	struct Libro p = {"Robert", "Algo-rithms"}; p.autor = "Robert"; q = p;

Abstractos	Clases	Datetime hoy = DateTime.Now;
	Listas	ArrayList legajos = new ArrayList();
	Pilas	Stack miPila = new Stack();
	Colas Árboles	Queue fila = new Queue();

C# - Tipos de Datos Primitivos

Nombre corto	Clase .NET	Tipo	Ancho intervalo (bits)	
Byte	Byte	Entero sin signo	8	0 a 255
Sbyte	Sbyte	Entero con signo	8	-128 a 127
Int	Int32	Entero con signo	32	-2.147.483.648 a 2.147.483.647
UInt	UInt32	Entero sin signo	32	0 a 4294967295
Short	Int16	Entero con signo	16	-32.768 a 32.767
Ushort	UInt16	Entero sin signo	16	0 a 65535
Long	Int64	Entero con signo	64	-9223372036854775508 a 9223372036854775507
Ulong	UInt64	Entero sin signo	64	0 a 18446744073709551615
Float	Single	Tipo de punto flotante de precisión simple	32	-3,402823e38 a 3,402823e38
double	Double	Tipo de punto flotante de precisión doble	64	1,79769313486232e308 a 1,79769313486232e308
char	Char	Un carácter Unicode	16	Símbolos Unicode utilizados en el texto
object	Object	Tipo base de todos los otros tipos		
string	String	Una secuencia de caracteres		
decimal	Decimal	Tipo preciso fraccionario o integral, que puede representar números decimales con 29 dígitos significativos	128	±1.0

Constantes

Las constantes son variables que no pueden cambiar de valor una vez que es establecido. Son valores inmutables que se conocen en tiempo de diseño (cuando se está diseñando el programa) y que no cambian durante la vida del programa.

	Un ejemplo para la definición

- Se debe anteponer el modificador *const*
- Por convención se escriben en mayúscula y las palabras se separan con guion bajo.

```
const int CANTIDAD_MESES = 12;

const double PI = 3.14159265359;

const string NOMBRE_SISTEMA = "Nyvus";
```

Comentarios de código

¿Qué son y para qué sirven?

Probablemente ya has experimentado los comentarios en algún código que hayas visto, sea C# o cualquier otro lenguaje de programación (el concepto de comentarios en código es universal). La forma en que son escritos varía bastante, así que vamos a ver algunos de los tipos de comentarios que puedes usar en tu código de C#.

Tipos de comentarios

1. Comentarios de una línea

El tipo de comentario más básico en C# es el comentario de una línea. Como su nombre indica, convierte una línea en un comentario.

Observemos cómo se vería en VsCode:

```
static void Main(string[] args)
{
    // Este es un comentario de una línea

}
```

Eso es todo: iniciá tus líneas con dos diagonales (//) y tu texto pasará de algo que el compilador revisa y se queja de ello, a algo que el compilador ignora completamente. Y ya que esto solo aplica a la línea con las diagonales, sos libre de hacer lo mismo en la línea siguiente, esencialmente usando comentarios de una línea para crear comentarios de múltiples líneas.

2. Comentarios con múltiples líneas

En caso de querer escribir comentarios de múltiples líneas, tiene más sentido usar la alternativa para comentarios de múltiples líneas que ofrece C#. En lugar de incluir el prefijo en cada línea, podés ingresar un carácter con secuencia al comienzo y al final (*start* y *stop*). Todo entre estos será tratado como comentarios.

```
static void Main(string[] args)
{
    /*  Este es todo un bloque de comentario
        ya que está encerrado entre
        barra asterisco de apertura
        y asterisco barra de cierre */

}
```

Usa la secuencia de inicio de diagonal-asterisco (/*), escribí lo que quieras, en varias líneas o no, y luego finalizá todo con la secuencia de finalización de asterisco-diagonal (*/). Entre esos marcadores, podés escribir lo que quieras.

Sentencias y bloques

En C#, una sentencia es una instrucción que se ejecuta en forma secuencial dentro de un bloque de código. Las sentencias son la unidad básica de ejecución en C#. Las sentencias pueden ser simples o compuestas y pueden incluir una o varias expresiones.

Ejemplos de **sentencias simples en C#** incluyen:

- Declaración de variables: *int* x = 10;
- Asignación de valores a variables: x = 20;
- Llamada a al método para mostrar por pantalla: *Console.WriteLine* ("Hola mundo!");
- Estructuras de control de flujo, como *if/else* y *switch* .

Las sentencias son otros de los elementos de construcción individuales que describen pasa-a-paso el cómo debe llevar a cabo un proceso que estemos programando, es decir, la descripción imperativa de lo que nuestro programa debe hacer.

Una sentencia es una unidad completa de ejecución y termina en punto y coma (;).

Como un programa es un conjunto de sentencias, siempre se termina cada línea con "; ".

Aunque hay algunas excepciones que veremos más adelante.

Bloques

Un bloque de código es un conjunto de cero o más sentencias agrupadas entre llaves {}. Los bloques de código se utilizan para agrupar sentencias relacionadas y permitir que se ejecuten juntas.

En VsCode se vería de esta forma:

```
static void Main(string[] args)
{
    {
        const int num1 = 1120;
        int num2 = 5;
        int resultado;
        resultado = num1 + num2;
        Console.WriteLine("El resultado es: " + resultado);
    }
}
```

Ha quedado claro el concepto de **sentencia**: es la unidad de construcción básica para especificar declaración de variables y de constantes, expresiones de asignación, ciclos repetitivos, sentencias de selección, etc.

Método Main

¿Cómo se utiliza en C#?

En C#, la función *Main()* es el punto de entrada de cualquier aplicación de consola. Es el primer método que se ejecuta cuando se inicia la aplicación y es el encargado de iniciar la ejecución del programa. El método *Main()* debe estar presente en cualquier aplicación de consola en C#.

La firma del método *Main()* debe ser la siguiente:

```
static void Main(string[] args)
```

- *static:* indica que el método pertenece a la clase y no a una instancia de la misma (veremos que significa más adelante).
- *void:* indica que el método no devuelve ningún valor (veremos que significa más adelante).
- *Main:* es el nombre del método.
- *string[] args:* es un vector o arreglo de cadenas que contiene los argumentos de línea de comandos que se pasan al programa al iniciarse. No los usaremos, pero tienen que estar siempre.

El método *Main()* puede contener cualquier código que se necesite para ejecutar la aplicación, como la creación de objetos, la llamada a métodos, la lectura de datos, la escritura de datos en la consola, etc.

Un ejemplo simple del método *Main()* podría ser el siguiente:

```
using System;

namespace ConsoleApp6
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            {
                Console.WriteLine("Hola mundo!");
                Console.ReadLine();
            }
        }
    }
}
```

En este ejemplo, el método *Main()* escribe "Hola mundo!" en la consola y espera a que el usuario presione *Enter* antes de finalizar la aplicación. La primera línea del método *Main()* utiliza la clase *Console* para escribir una cadena en la consola. La segunda línea utiliza la misma clase para leer una línea de texto del usuario.

Mostrar datos por pantalla

¿Cómo lo hacemos?

Console.WriteLine() es un método de la clase *Console* en C# que se utiliza para imprimir texto en la consola de la aplicación. El método *WriteLine()* es una sobrecarga del método *Write()* y agrega una nueva línea al final de la cadena de texto que se va a imprimir en la consola.

La sintaxis del método *Console.WriteLine()*:

Console.WriteLine(valor);

```
Console.WriteLine("Hola mundo!");
```

El parámetro valor es el valor que se va a imprimir en la consola. Este puede ser una cadena de texto, una variable, una expresión, o cualquier otra cosa que se pueda convertir en una cadena.

Ingreso de datos por teclado

¿Cómo lo utilizamos?

Console.ReadLine() es un método de la clase *Console* en C# que se utiliza para leer una línea de entrada del usuario desde la consola. El método *ReadLine()* espera hasta que el usuario escriba una línea de texto en la consola y presione la tecla *Enter*.

La sintaxis del método *Console.ReadLine()* es la siguiente:

```
string entrada = Console.ReadLine();
```

El método *Console.ReadLine()* **SIEMPRE** devuelve una cadena de texto que representa la línea de entrada que se leyó desde la consola. La cadena de texto devuelta incluye cualquier carácter que se escriba en la línea de entrada hasta que se presione *Enter*.

Por ejemplo, el siguiente código utiliza el método *Console.ReadLine()* para leer una línea de texto ingresada por el usuario y luego la imprime en la consola:

```
static void Main(string[] args)
{
    {
        Console.WriteLine("Escribe una línea de texto:");
        string entrada = Console.ReadLine();
        Console.WriteLine("La línea de entrada fue: " + entrada);
    }
}
```

Cuando se ejecuta el código, la aplicación espera a que el usuario escriba una línea de texto en la consola y presione *Enter*. Luego, la aplicación almacena la línea de entrada en la variable entrada y la imprime en la consola utilizando el método *Console.WriteLine()*.

Es importante tener en cuenta que el método *Console.ReadLine()* bloquea la ejecución del programa hasta que el usuario ingrese una línea de texto en la consola. Por lo tanto, si se llama al método *Console.ReadLine()* en el método *Main()*, la aplicación se quedará esperando la entrada del usuario y no continuará ejecutando ninguna línea de código después de esa llamada hasta que el usuario ingrese una línea de texto en la consola.

¿Cómo u tilizar el comando Parse?

Parse es un método utilizado en muchos lenguajes de programación para convertir una cadena de texto en un tipo de dato específico , como un número entero, un número decimal, un booleano, etc. El método *Parse* toma una cadena de texto como entrada y la convierte en un valor del tipo de dato especificado.

En C#, por ejemplo, se pueden utilizar los métodos *int.Parse()* y *double.Parse()* para convertir una cadena de texto en un número entero o un número decimal, respectivamente. Si la cadena de texto no se puede convertir en el tipo de dato especificado, el método *Parse* generará una excepción.

Por ejemplo, en el siguiente código, se utiliza el método *int.Parse()* para convertir la cadena de texto "10" en un valor entero:

```
static void Main(string[] args)
{
    {
        string texto = "10";
        int numero = int.Parse(texto);
    }
}
```

En este caso, el valor de la variable `numero` sería 10. Si la variable `texto` contuviera una cadena de texto que no se pudiera convertir en un número entero, como por ejemplo "abc", el método *`int.Parse()`* generaría una excepción *`FormatException`* `(Error en tiempo de ejecución).`

En resumen, *`Parse`* es un método utilizado para convertir una cadena de texto en un tipo de dato específico en lenguajes de programación como C#.

Ingreso de un valor numérico

¿Cómo lo utilizamos?

Para ingresar un entero desde la consola en C#, se puede utilizar el método *Console.ReadLine()* para leer una cadena de texto desde la consola y luego convertir esa cadena a un valor entero utilizando el método *int.Parse()* o *int.TryParse()*.

La sintaxis para leer una línea de entrada de la consola y convertirla a un valor entero es la siguiente:

```
static void Main(string[] args)
{
    {
        string entrada;
        int numero;
        Console.WriteLine("Ingresa un número entero:");
        entrada = Console.ReadLine();
        numero = int.Parse(entrada);
        Console.WriteLine("El numero ingresado es: " + numero);
    }
}
```

En el ejemplo, vemos como la aplicación imprime un mensaje en la consola pidiéndole al usuario que ingrese un número entero. Luego, utiliza el método *Console.ReadLine()* para leer una línea de entrada de la consola y almacenarla en la variable entrada.

Finalmente, utiliza el método *int.Parse()* para convertir la cadena de texto almacenada en entrada a un valor entero y lo almacena en la variable numero y muestra el valor ingresado por pantalla .

Si el usuario/a ingresa una cadena que no se puede convertir a un valor entero, el método *int.Parse()* generará un error de excepción *FormatException*.

Otra forma sería:

```
static void Main(string[] args)
{
    {
        int numero;
        Console.WriteLine("Ingresa un número entero:");
        numero = int.Parse(Console.ReadLine());
        Console.WriteLine("El numero ingresado es: " + numero);
    }
}
```

En nuestro curso, asumimos por el momento que el usuario/a no se va a equivocar en el tipo de datos a ingresar.

Aunque si quisiéramos validarlo, también es posible utilizar el método *int.TryParse()* para convertir la cadena de texto a un valor entero de forma más segura. Este método intenta realizar la conversión y devuelve true si la conversión es exitosa y false si la conversión falla. En caso de éxito, el valor convertido se almacena en una variable de salida proporcionada. La sintaxis para el método *int.TryParse()* es la siguiente:

```
static void Main(string[] args)
{
    {
        Console.WriteLine("Ingresa un número entero:");
        string entrada = Console.ReadLine();
        int numero;
        if (int.TryParse(entrada, out numero))
        {
            Console.WriteLine("El número ingresado es: " + numero);
        }
        else
        {
            Console.WriteLine("La entrada no es un número entero válido.");
        }
    }
}
```

En el ejemplo, el método *int.TryParse()* intenta convertir la cadena de texto entrada a un valor entero y lo almacena en la variable numero. Si la conversión es exitosa, el método devuelve *true* y la aplicación imprime el valor convertido en la consola. Si la conversión falla, el método devuelve *false* y la aplicación imprime un mensaje de error en la consola.

Al igual que para leer un valor entero se debe "parsear" el dato leído, para un *double* sería:

```
static void Main(string[] args)
{
    {
        double numero;
        Console.WriteLine("Ingresa un número decimal:");
        numero = double.Parse(Console.ReadLine());
        Console.WriteLine("El número ingresado es: " + numero);
    }
}
```