Bienvenido a C#

C# es un elegante lenguaje orientado a objetos que permite a los desarrolladores crear una variedad de aplicaciones seguras y sólidas que se ejecutan en **.NET Framework.**
Puede usar C# para crear aplicaciones de Windows, servicios web, aplicaciones móviles, aplicaciones cliente-servidor, aplicaciones de bases de datos y mucho, mucho más.

El Framework .NET

.NET Framework consta de **Common Language Runtime (CLR)** y la **biblioteca de clases** de .NET Framework .
El **CLR** es la base de .NET Framework. Administra el código en el momento de la ejecución, brindando servicios básicos como la administración de la memoria, la precisión del código y muchos otros aspectos de su código.
La **biblioteca de clases** es una colección de clases, interfaces y tipos de valores que le permiten realizar una variedad de tareas de programación comunes, como la recopilación de datos, el acceso a archivos y el trabajo con texto.
Los programas de C# usan la biblioteca de clases de .NET Framework ampliamente para realizar tareas comunes y proporcionar diversas funcionalidades.

Variables

Los programas suelen utilizar datos para realizar tareas.
La creación de una **variable** reserva una ubicación de memoria, o un espacio en la memoria, para almacenar valores. Se llama **variable** porque la información almacenada en esa ubicación se puede cambiar cuando el programa se está ejecutando.
Para usar una variable, primero debe declararse especificando el **nombre** y **el tipo de datos** .
Un nombre de variable, también llamado **identificador** , puede contener letras, números y el carácter de subrayado (_) y debe comenzar con una letra o subrayado.
Aunque el nombre de una variable puede ser cualquier conjunto de letras y números, el mejor identificador es **descriptivo** de los datos que contendrá. ¡Esto es muy importante para crear un código claro, comprensible y legible!

Tipos de variables

Un **tipo de datos** define la información que se puede almacenar en una variable, el tamaño de la memoria necesaria y las operaciones que se pueden realizar con la variable.
Por ejemplo, para almacenar un valor entero (un número entero) en una variable, use la palabra clave **int** :

```
int myAge;
```
Copiar

El código anterior declara una variable llamada **myAge** de tipo **entero** .

Tipos de datos integrados

Hay una serie de tipos de datos integrados en C#. Los más comunes son:
int - entero.
float - número de punto flotante (decimales).
doble - versión de doble precisión del flotador.
char - un solo carácter.
bool : valor booleano que solo puede tener uno de dos valores: verdadero o falso.
string - una secuencia de caracteres.

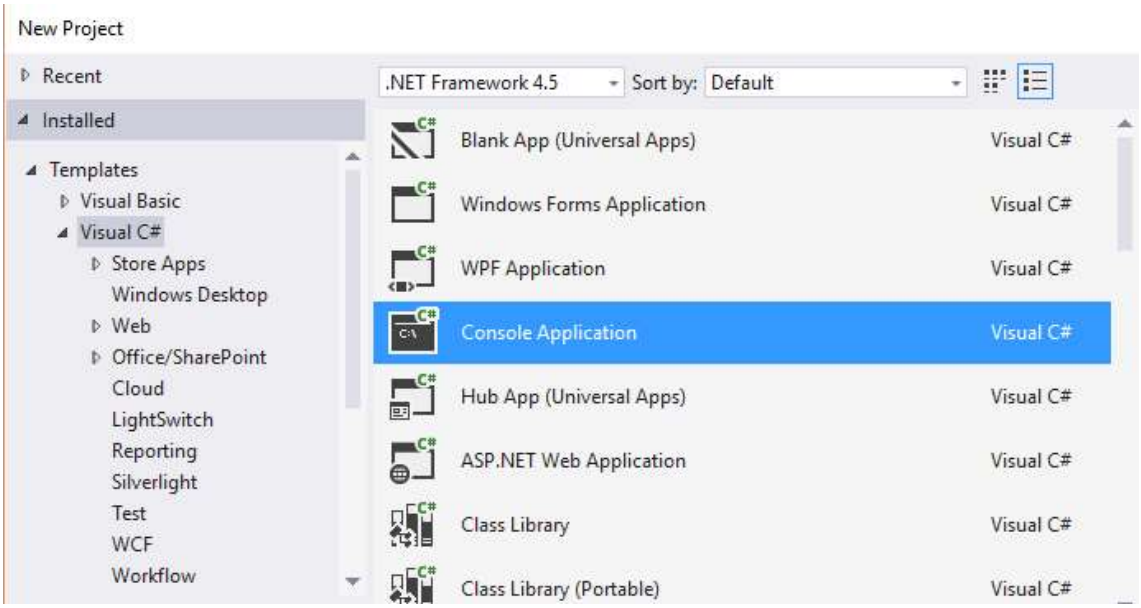
Las siguientes declaraciones usan tipos de datos de C#:

```
int x = 42;
double pi = 3.14;
char y = 'Z';
bool isOnline = true;
string firstName = "David";
```
Copiar

Tenga en cuenta que los valores de **char** se asignan mediante comillas simples y los valores de **string** requieren comillas dobles.

Su primer programa C#

Para crear un programa C#, debe instalar un entorno de desarrollo integrado (IDE) con herramientas de codificación y depuración.
Usaremos **Visual Studio Community Edition** , que está disponible para descargar de forma gratuita.
Después de instalarlo, elija la configuración predeterminada.
A continuación, haga clic en **Archivo->Nuevo->Proyecto** y luego seleccione **Aplicación de consola** como se muestra a continuación:



Introduzca un nombre para su proyecto y haga clic en Aceptar.

La aplicación de consola utiliza una interfaz de solo texto. Elegimos este tipo de aplicación para centrarnos en aprender los fundamentos de C#.

Visual Studio generará automáticamente algo de código para su proyecto:

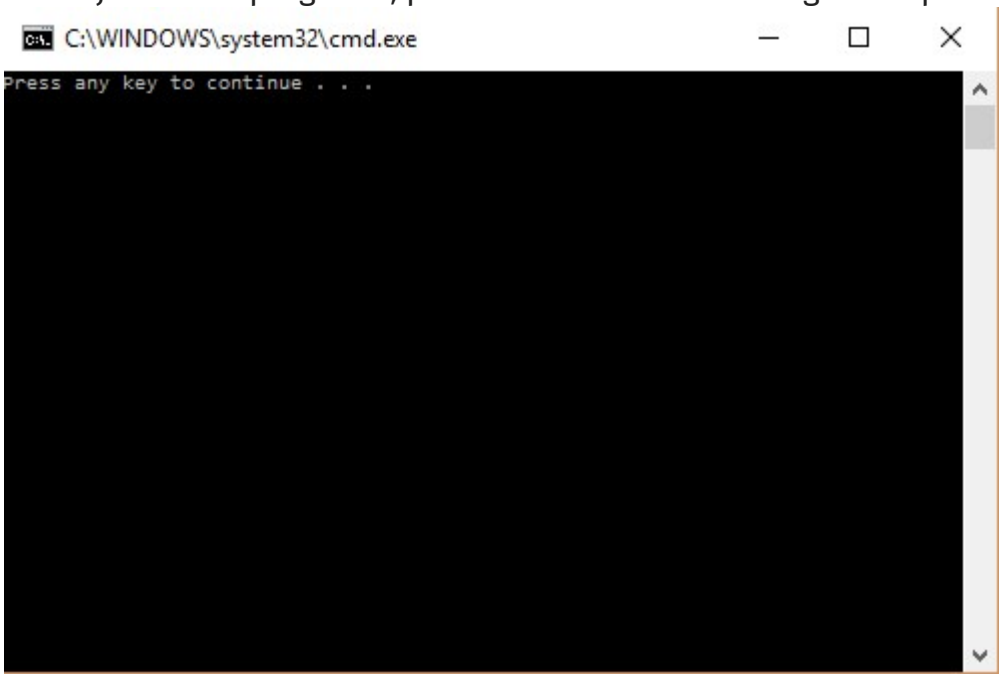
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SoloLearn
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

Copiar

Por ahora, recuerde que cada aplicación de consola de C# debe contener un **método (una función) denominado Main** . Main es el punto de partida de toda aplicación, es decir, el punto desde el que se inicia la ejecución de nuestro programa.

Para ejecutar su programa, presione **Ctrl+F5** . Verá la siguiente pantalla:



Esta es una ventana de consola. Como no teníamos declaraciones en nuestro método **principal** , el programa solo produce un mensaje general. Presionar cualquier tecla cerrará la consola.

Visualización de salida

La mayoría de las aplicaciones requieren alguna **entrada** por parte del usuario y dan **salida** como resultado.
Para mostrar texto en la ventana de la consola, utilice los métodos **Console.Write** o **Console.WriteLine** . La diferencia entre estos dos es que **Console.WriteLine** va seguido de un terminador de línea, que mueve el cursor a la siguiente línea después de la salida del texto.

El siguiente programa mostrará Hello World! a la ventana de la consola:

```
static void Main(string[] args)
{
    Console.WriteLine("Hello World!");
}
```

Haga clic para ejecutar

Tenga en cuenta los **paréntesis** después del método **WriteLine** . Esta es la forma de pasar datos o argumentos a los métodos. En nuestro caso, **WriteLine** es el método y pasamos "Hello World!" a ella como argumento. Los argumentos de cadena deben ir entre comillas.

Podemos mostrar valores de variables en la ventana de la consola:

```
static void Main(string[] args)
{
    int x = 89;
    Console.WriteLine(x);
}
// Outputs 89
```
Haga clic para ejecutar

Para mostrar una **cadena con formato** , utilice la siguiente sintaxis:

```
static void Main(string[] args)
{
    int x = 10;
    double y = 20;

    Console.WriteLine("x = {0}; y = {1}", x, y);
}
```
Haga clic para ejecutar

Como puede ver, el valor de **x** reemplazó a **{0}** y el valor de **y** reemplazó **a {1}** .

Puede tener tantos marcadores de posición variables como necesite. (es decir: {3}, {4}, etc.).

Entrada del usuario

También puede solicitar al usuario que ingrese datos y luego usar el método **Console.ReadLine** para asignar la entrada a una variable de cadena. El siguiente ejemplo le pide al usuario un nombre y luego muestra un mensaje que incluye la entrada:

```
static void Main(string[] args)
{
    string yourName;
    Console.WriteLine("What is your name?");

    yourName = Console.ReadLine();

    Console.WriteLine("Hello {0}", yourName);
}
```
Haga clic para ejecutar

El método **Console.ReadLine** espera la entrada del usuario y luego la asigna a la variable. La siguiente declaración muestra una cadena formateada que contiene Hola con la entrada del usuario. Por ejemplo, si ingresa David, la salida será Hola David.
El método **Console.ReadLine()** devuelve un valor string.
Si espera otro tipo de valor (como int o double), los datos ingresados deben

convertirse a ese tipo.
Esto se puede hacer usando los métodos **Convert.ToXXX** , donde XXX es el nombre .NET del tipo al que queremos convertir. Por ejemplo, los métodos incluyen **Convert.ToDouble** y **Convert.ToBoolean** .
Para la conversión de enteros, hay tres alternativas disponibles según el tamaño de bits del
entero: **Convert.ToInt16** , **Convert.ToInt32** y **Convert.ToInt64** . El tipo int predeterminado en C# es de 32 bits.
Vamos a crear un programa que tome un número entero como entrada y lo muestre en un mensaje:

```
static void Main(string[] args)
{
  int age = Convert.ToInt32(Console.ReadLine());
  Console.WriteLine("You are {0} years old", age);
}
```

Haga clic para ejecutar

Si, en el programa anterior, se ingresa un valor no entero (por ejemplo, letras), la **conversión** fallará y provocará un error.

Comentarios

Los comentarios son declaraciones explicativas que puede incluir en un programa para beneficiar al lector de su código.
El compilador ignora todo lo que aparece en el comentario, por lo que nada de esa información afecta el resultado.

Un comentario que comienza con dos barras (//) se denomina comentario de una sola línea. Las barras le dicen al compilador que ignore todo lo que sigue, hasta el final de la línea.

```
// Prints Hello
Console.WriteLine("Hello");
```

Haga clic para ejecutar

Cuando ejecute este código, Hello se mostrará en la pantalla. La línea // Prints Hello es un comentario y no aparecerá como salida.

Los comentarios que requieren varias líneas comienzan con /* y terminan con */ al final del bloque de comentarios.
Puede colocarlos en la misma línea o insertar una o más líneas entre ellos.

```
/* Some long
   comment text
*/
int x = 42;
Console.WriteLine(x);
```

Haga clic para ejecutar

Agregar comentarios a su código es una buena práctica de programación. Facilita una comprensión clara del código para usted y para otros que lo lean.

La palabra clave var

Una variable se puede declarar explícitamente con su **tipo** antes de usarla. Como alternativa, C# proporciona una función útil para permitir que el compilador determine el tipo de la variable automáticamente en función de la expresión a la que está asignada.

La palabra clave var se usa para esos escenarios:

```
var num = 15;
```
Copiar

El código anterior hace que el compilador determine el tipo de variable. Dado que el valor asignado a la variable es un número entero, la variable se declarará automáticamente como un número entero.

Las variables declaradas usando la palabra clave **var** se denominan variables **implícitamente tipadas** .

Las variables tipificadas implícitamente **deben** inicializarse con un valor.

Por ejemplo, el siguiente programa generará un error:

```
var num;
num = 42;
```
Copiar

Aunque es fácil y conveniente declarar variables usando la palabra clave **var** , el uso excesivo puede dañar la legibilidad de su código. La mejor práctica es declarar explícitamente las variables.

constantes

Las constantes almacenan un valor que no se puede cambiar desde su asignación inicial.

Para declarar una constante, use el modificador **const** .

Por ejemplo:

```
const double PI = 3.14;
```
Copiar

El valor de const PI no se puede cambiar durante la ejecución del programa. Por ejemplo, una instrucción de asignación más adelante en el programa generará un error:

```
const double PI = 3.14;
PI = 8; //error
```
Copiar

Las constantes **deben** inicializarse con un valor cuando se declaran.

Operadores

Un **operador** es un símbolo que realiza manipulaciones matemáticas o lógicas.

Operadores aritméticos

C# admite los siguientes operadores aritméticos:

Operator	Symbol	Form
Addition	+	x + y
Subtraction	-	x - y
Multiplication	*	x * y
Division	/	x / y
Modulus	%	x % y

Por ejemplo:

```
int x = 10;
int y = 4;
Console.WriteLine(x-y);
```

Haga clic para ejecutar

Division

El operador de división (/) divide el primer operando por el segundo. Si los operandos son ambos enteros, cualquier resto se elimina para devolver un valor entero.

Ejemplo:

```
int x = 10 / 4;
Console.WriteLine(x);
```

Haga clic para ejecutar

La división por 0 no está definida y bloqueará su programa.

Módulo

El operador de módulo (%) se conoce informalmente como operador de resto porque devuelve el resto de una división entera.

Por ejemplo:

```
int x = 25 % 7;
Console.WriteLine(x);
```

Precedencia de operadores

La precedencia de operadores determina la agrupación de términos en una expresión, lo que afecta la forma en que se evalúa una expresión. Ciertos operadores tienen mayor prioridad sobre otros; por ejemplo, el operador de multiplicación tiene mayor precedencia que el operador de suma.
Por ejemplo:

```
int x = 4+3*2;
Console.WriteLine(x);
```

Haga clic para ejecutar

El programa evalúa primero 3*2 y luego suma el resultado a 4.
Como en matemáticas, el uso **de paréntesis** altera la precedencia de los operadores.

```
int x = (4 + 3) *2;
Console.WriteLine(x);
```

Haga clic para ejecutar

Las operaciones entre paréntesis se realizan primero. Si hay expresiones entre paréntesis anidadas unas dentro de otras, la expresión dentro de los paréntesis más internos se evalúa primero.

Si ninguna de las expresiones está entre paréntesis, los operadores multiplicativos (multiplicación, división, módulo) se evaluarán antes que los operadores aditivos (suma, resta). Los operadores de igual precedencia se evalúan de izquierda a derecha.

Operadores de Asignación

El operador **de asignación** = asigna el valor del lado derecho del operador a la variable del lado izquierdo.

C# también proporciona **operadores de asignación compuestos** que realizan una operación y una asignación en una instrucción.
Por ejemplo:

```
int x = 42;
x += 2; // equivalent to x = x + 2
x -= 6; // equivalent to x = x - 6
```

La misma sintaxis abreviada se aplica a los operadores de multiplicación, división y módulo.

```
x *= 8; // equivalent to x = x * 8
x /= 5; // equivalent to x = x / 5
x %= 2; // equivalent to x = x % 2
```

Copiar

Operador de incremento

El operador **de incremento** se usa para aumentar el valor de un número entero en uno y es un operador de C# de uso común.

```
x++; //equivalent to x = x + 1
```

Copiar

Por ejemplo:

```
int x = 10;
x++;
Console.WriteLine(x);
```

Mostrara por consola el valor 11.

Formas de prefijo y posfijo

El operador de incremento tiene dos formas, **prefijo** y **posfijo** .

```
++x; //prefix
x++; //postfix
```

Copiar

Prefijo incrementa el valor y luego continúa con la expresión.
Postfix evalúa la expresión y luego realiza el incremento.

Ejemplo de prefijo:

```
int x = 3;
int y = ++x;
// x is 4, y is 4
```

Haga clic para ejecutar

Ejemplo de sufijo:

```
int x = 3;
int y = x++;
// x is 4, y is 3
```

Haga clic para ejecutar

El ejemplo **del prefijo** incrementa el valor de x y luego lo asigna a y.
El ejemplo **del sufijo** asigna el valor de x a y, y luego incrementa x.

Operador de decremento

El operador **de decremento** (--) funciona de la misma manera que el operador de incremento, pero en lugar de aumentar el valor, lo disminuye en uno.

```
--x;  // prefix
x--;  // postfix
```
Copiar

El operador de decremento (--) funciona de la misma manera que el operador de incremento.