Manual C#

Indice

¿Que es C#?

Que es C#

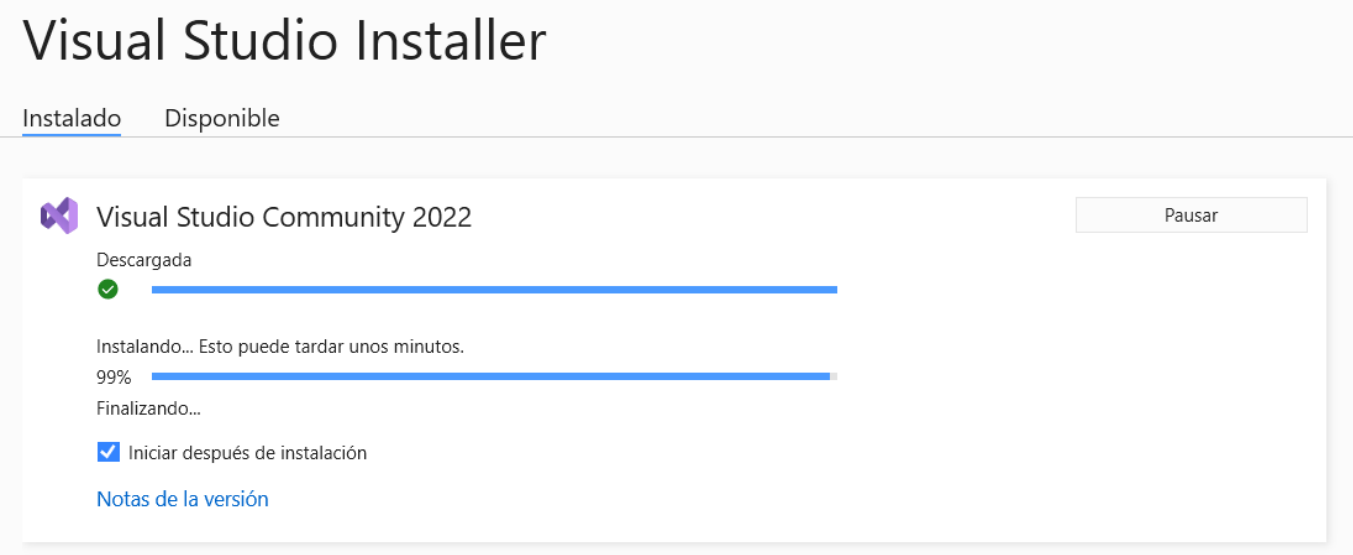
C# es un lenguaje de programacion creado por Microsoft orientado a objetos que utiliza .NET Framework.
Es de la familia de C. Es parecido a C++ y a JAVA.

La primera version fue lanzada en 2002. La ultima es la 12, lanzada en noviembre de 2023.
\newpage

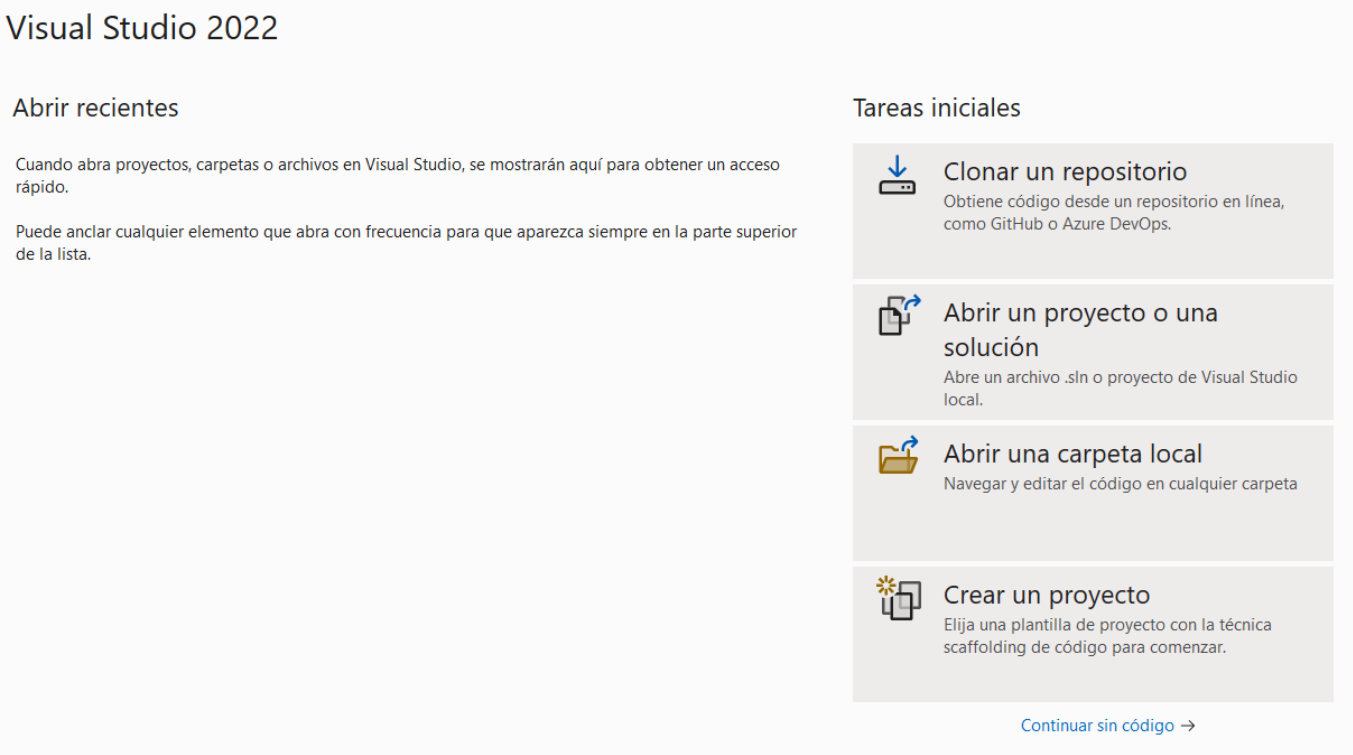
Instalar C# IDE en Windows

Instalar C#IDE en windows

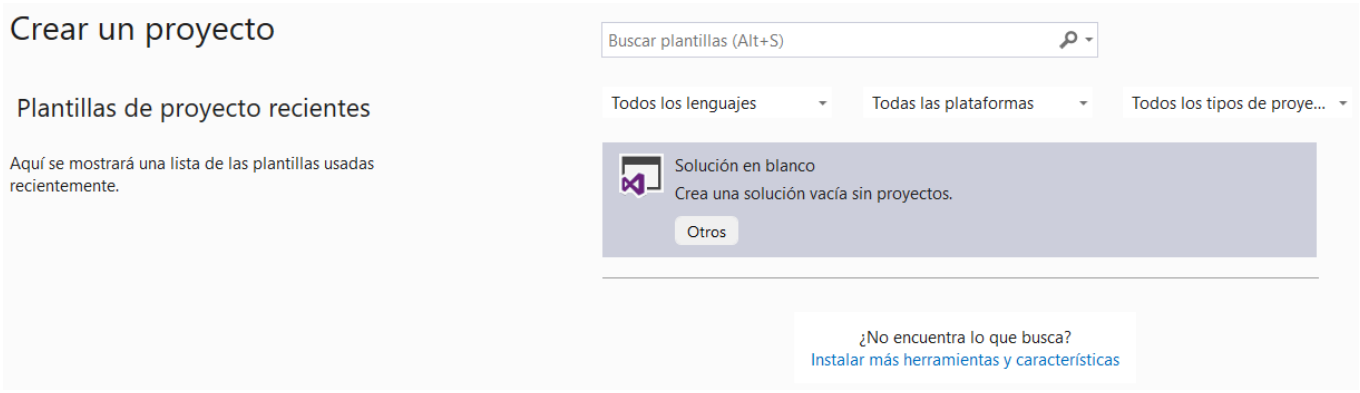
Lo primero es instalar .NET. https://visualstudio.microsoft.com/es/vs/community/



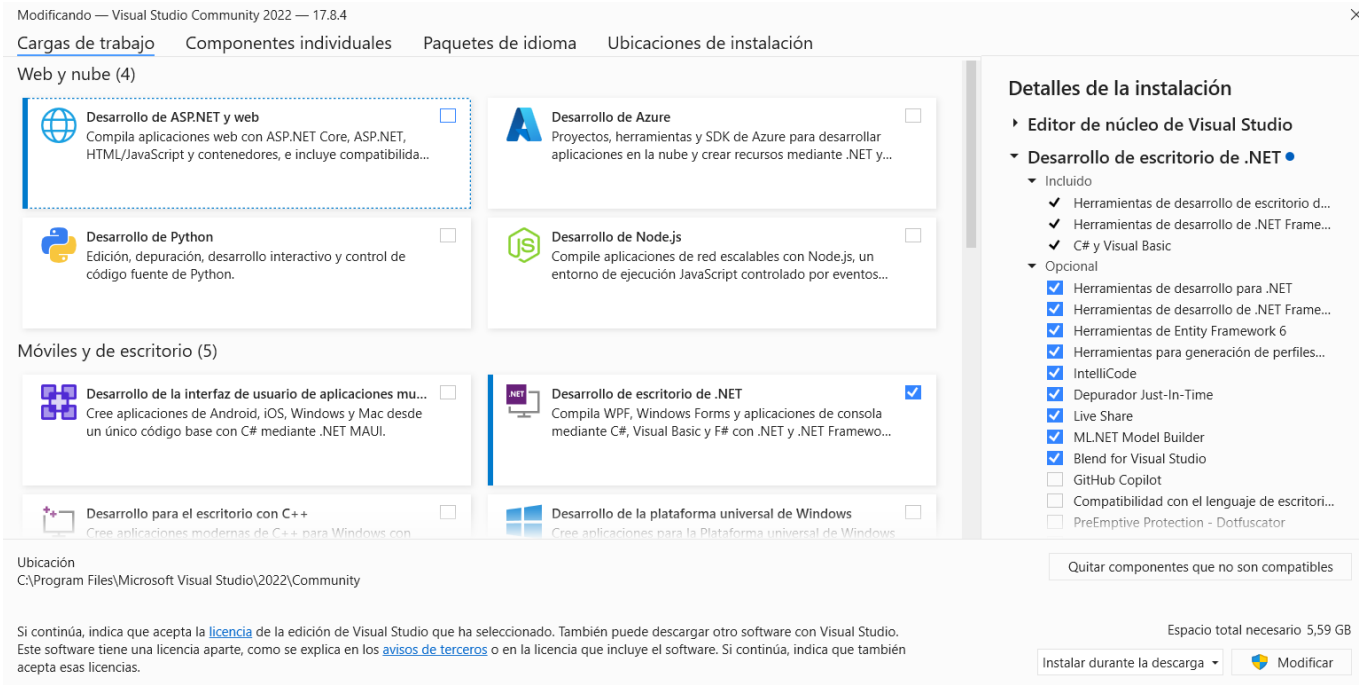
Luego creamos proyecto



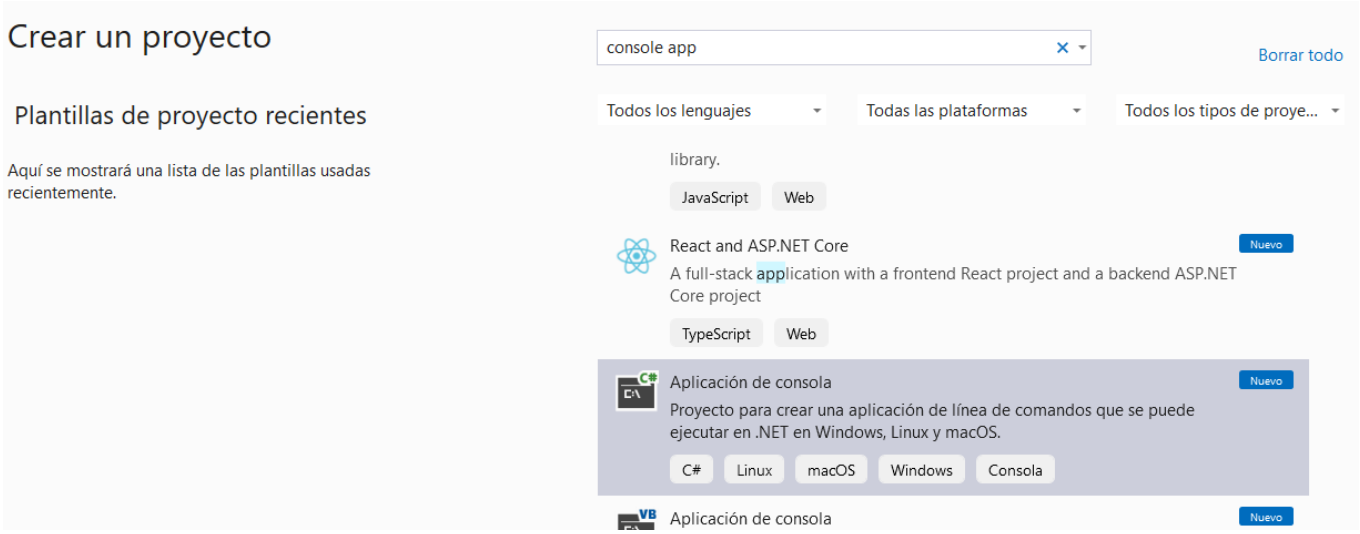
Instalamos mas herramietas y caracteristicas.



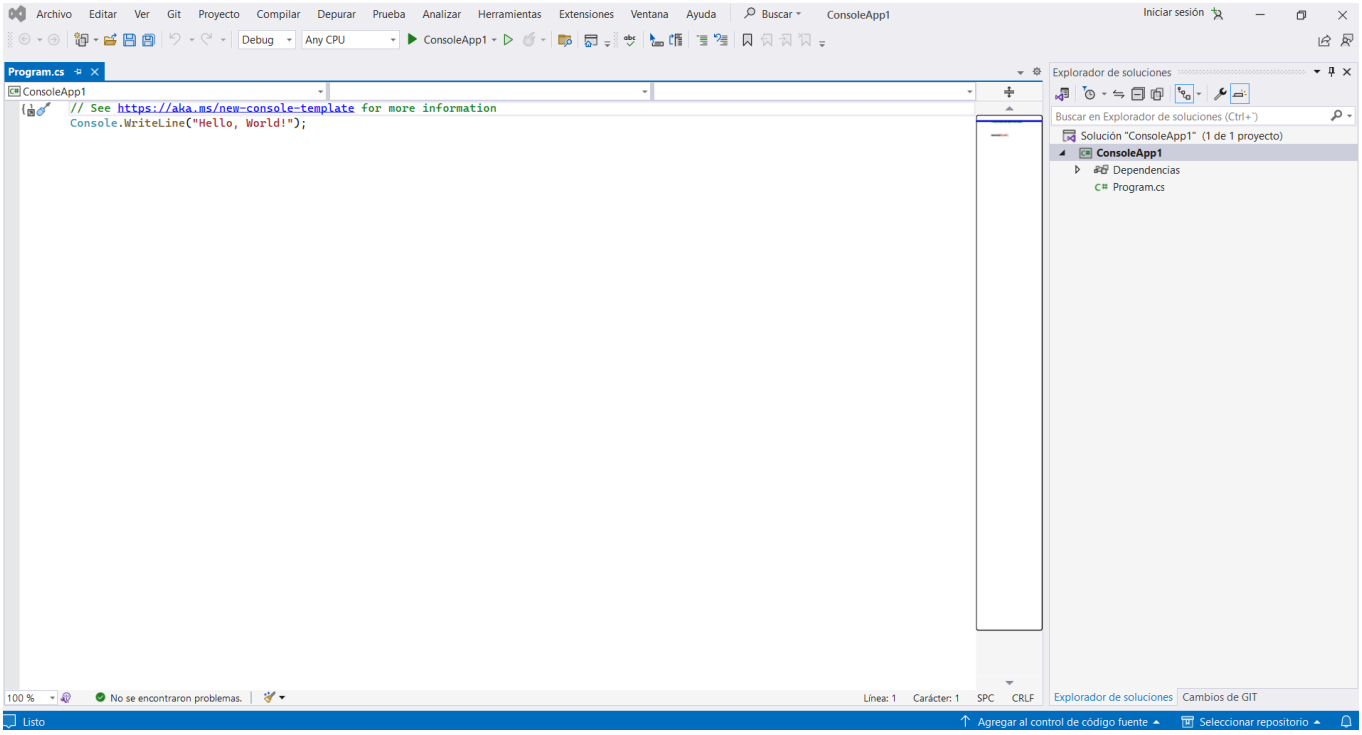
Instalamos desarrollo de escritorio .NET



En crear proyecto elegimos Aplicacion de consola



Le damos a continuar y ya lo tenemos


Archivo Editar Ver Git Proyecto Compilar Depurar Prueba Analizar Herramientas Extensiones Ventana Ayuda
Buscar
ConsoleApp1
Iniciar sesión
Debug
Any CPU
ConsoleApp1
Program.cs
ConsoleApp1
// See https://aka.ms/new-console-template for more information
Console.WriteLine("Hello, World!");
Explorador de soluciones
Buscar en Explorador de soluciones (Ctrl+')
Solución "ConsoleApp1" (1 de 1 proyecto)
ConsoleApp1
Dependencias
Program.cs
100 %
No se encontraron problemas.
Línea: 1
Carácter: 1
SPC
CRLF
Explorador de soluciones
Cambios de GIT
Listo
Agregar al control de código fuente
Seleccionar repositorio

Instalar C# en Linux

Instalar C# en Linux

Descargar el script con wget:

```
wget https://dot.net/v1/dotnet-install.sh -O dotnet-install.sh
```

Darle permiso de ejecucion el script:

```
chmod +x ./dotnet-install.sh
```

Ejecutamos con el parametro --version latest para instalar la ultima version:

```
./dotnet-install.sh --version latest
```

Ponemos las variables de entorno:

```
export DOTNET_ROOT=$HOME/.dotnet
```

```
export PATH=$PATH:$DOTNET_ROOT:$DOTNET_ROOT/tools
```

Creacion de Hola mundo

Creacion de Hola mundo

Escribir en el terminal este comando:

```
dotnet new console --framework net6.0 --use-program-main
```

Crear archivo Program.cs y poner este codigo:

```
namespace HelloWorld;

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

Escribe `dotnet run` en la consola para ejecutarlo:

```
dotnet run
Hello, World!
```

Explicacion del codigo

Linea 1: `using System` significa que podemos usar clases del namespace System.

Linea 2: Linea en blanco. C# ignora las ignora.

Linea 3: `namespace` se usa para organizar el codigo.

Linea 4: Las llaves `{}` marcan el inicio y el fin del bloque de codigo.

Linea 5: `class` es un contenedor de datos y métodos, que aporta funcionalidad al programa. Cada línea de código que se ejecuta en C# debe estar dentro de una clase. En el ejemplo, llamamos a la clase `Program`.

Linea 7: Otra cosa que siempre aparece en un programa C# es el método `Main`. Se ejecutará cualquier código dentro de las llaves `{}`.

Linea 9: `Console` es una clase del espacio de nombres `System`, que tiene un método `WriteLine()` que se utiliza para imprimir texto. En el ejemplo, aparece "¡Hello World!".

Salidas

Salidas

Para generar valores o imprimir texto en C#, puede utilizar el método `WriteLine()`:

```
Console.WriteLine("Hello World!");
```

Puede agregar tantos métodos `WriteLine()` como quieras. Ten en cuenta que aparecera una nueva línea para cada método:

```
Console.WriteLine("Hello World!");
Console.WriteLine("I am Learning C#");
Console.WriteLine("It is awesome!");
```

También puedes generar números y realizar cálculos matemáticos:

```
Console.WriteLine(3 + 3);
```

El metodo Write

También existe un método `Write()`, que es similar a `WriteLine()`.

La única diferencia es que no inserta una nueva línea al final de la salida:

```
Console.Write("Hello World! ");
Console.Write("I will print on the same line.");
```

Comentarios

Comentarios

Los comentarios se pueden utilizar para explicar el código C# y hacerlo más legible. También se puede utilizar para evitar la ejecución al probar código alternativo.

Comentarios de una linea

Los comentarios de una linea comienzan con dos barras diagonales `//`.

C# ignora cualquier texto entre `//` y el final de la línea (no se ejecutará):

```
// This is a comment
Console.WriteLine("Hello World!");
```

Este ejemplo utiliza un comentario de una linea al final de una linea de código:

```
Console.WriteLine("Hello World!");  // This is a comment
```

Comentarios de varias líneas

Comentarios de varias líneas

Los comentarios de varias líneas comienzan con `/*` y terminan con `*/`

C# ignorará cualquier texto entre `/*` y `*/`

Este ejemplo utiliza un comentario de varias líneas para explicar el código:

```
/* The code below will print the words Hello World
to the screen, and it is amazing */
Console.WriteLine("Hello World!");
```

Variables

Variables

Las variables son contenedores para almacenar valores de datos.

En C#, existen diferentes tipos de variables, por ejemplo:

- `int` - almacena números enteros, sin decimales, como 123 o -123
- `double` - almacena números de coma flotante, con decimales, como 19,99 o -19,99
- `char` - almacena caracteres individuales, como 'a' o 'B'. Los valores de caracteres están entre comillas simples.
- `string` - almacena texto, como "Hola mundo". Los valores de cadena están entre comillas dobles.
- `bool` - almacena valores con dos estados: verdadero o falso

Declaración de variables

Para crear una variable, debe especificar el tipo y asignarle un valor:

```
type variableName = value;
```

Donde `type` es un tipo de C# (como `int` o `string`) y `variableName` es el nombre de la variable (como x o nombre). El signo igual se utiliza para asignar valores a la variable.

Para crear una variable que debería almacenar texto: Creamos una variable llamada `name` de tipo `string` y le asignamos el valor "John":

```
string name = "John";
Console.WriteLine(name);
```

Para crear una variable que debería almacenar un número: Creamos una variable llamada `myNum` de tipo `int` y le asignamos el valor 15 :

```
int myNum = 15;
Console.WriteLine(myNum);
```

También puedes declarar una variable sin asignar el valor y asignar el valor más tarde:

```
int myNum;
myNum = 15;
Console.WriteLine(myNum);
```

Ten en cuenta que si asignamos un nuevo valor a una variable existente, sobrescribirá el valor anterior:

```
int myNum = 15;
myNum = 20; // myNum is now 20
Console.WriteLine(myNum);
```

Otros tipos

```
int myNum = 5;
double myDoubleNum = 5.99D;
char myLetter = 'D';
bool myBool = true;
string myText = "Hello";
```

Constantes

Constantes

Si no quieres que se sobrescriban los valores existentes, puede agregar la palabra clave `const` delante del tipo de variable.

Esto declarará la variable como "constante", lo que significa que no se puede cambiar y es de solo lectura:

```
const int myNum = 15;
myNum = 20; // error
```

La palabra clave `const` es útil cuando quieres que una variable almacene siempre el mismo valor, para no estropear el código. Un ejemplo que a menudo se hace referencia como constante es PI (3,14159...).

Nota: No se puede declarar una variable constante sin asignar el valor. Si lo haces, se producirá un error: un campo constante requiere que se proporcione un valor .

Variables de visualización

Variables de visualización

Mostrar variables

El método `WriteLine()` se utiliza a menudo para mostrar valores de variables en la ventana de la consola.

Para combinar texto y una variable, use el `+` carácter:

```
string name = "John";
Console.WriteLine("Hello " + name);
```

También puedes usar el carácter `+` para agregar una variable a otra variable:

```
string firstName = "John ";
string lastName = "Doe";
string fullName = firstName + lastName;
Console.WriteLine(fullName);
```

Para valores numéricos, el carácter `+` funciona como un operador matemático:

```
int x = 5;
int y = 6;
Console.WriteLine(x + y); // Print the value of x + y
```

Múltiples variables

Múltiples variables

Declarar muchas variables

Para declarar más de una variable del mismo tipo , utilice una lista separada por comas:

```
int x = 5, y = 6, z = 50;
Console.WriteLine(x + y + z);
```

También puedes asignar el mismo valor a varias variables en una línea:

```
int x, y, z;
x = y = z = 50;
Console.WriteLine(x + y + z);
```

Identificadores

Identificadores

Todas las variables de C# deben identificarse con nombres únicos .

Estos nombres únicos se denominan identificadores .

Los identificadores pueden ser nombres cortos (como x o y) o nombres más descriptivos (age, sum, totalVolume).

Nota: Se recomienda utilizar nombres descriptivos para crear código comprensible y mantenible:

```
// Bien
int minutesPerHour = 60;

// OK, pero no es facil de entender
int m = 60;
```

Las reglas generales para nombrar variables son:

- Los nombres pueden contener letras, dígitos y el carácter de subrayado
- Los nombres deben comenzar con una letra o un guión bajo.
- Los nombres deben comenzar con una letra minúscula y no pueden contener espacios en blanco.
- Los nombres distinguen entre mayúsculas y minúsculas ("myVar" y "myvar" son variables diferentes).
- Las palabras reservadas (como palabras clave de C#, como `int` o `double`) no se pueden utilizar como nombres.

Tipos de datos

Tipos de datos

Como se explica en el capítulo de variables, una variable en C# debe ser un tipo de datos específico:

```
int myNum = 5;               // Integer
double myDoubleNum = 5.99D;  // Floating point number
char myLetter = 'D';         // Character
bool myBool = true;          // Boolean
string myText = "Hello";     // String
```

Un tipo de datos especifica el tamaño y el tipo de valores de variables.

Es importante utilizar el tipo de datos correcto para la variable correspondiente; para evitar errores, ahorrar tiempo y memoria, pero también hará que su código sea más fácil de mantener y legible. Los tipos de datos más comunes son:

Tipo de dato	Tamaño	Descripcion
int	4 bytes	Almacena numeros enteros desde -2,147,483,648 a 2,147,483,647
long	8 bytes	Almacena numero enteros desde -9,223,372,036,854,775,808 a 9,223,372,036,854,775,807
float	4 bytes	Almacena numeros fraccionales suficiente para almacenar de 6 a 7 dígitos decimales
double	8 bytes	Almacena numeros fraccinonales Suficiente para almacenar 15 dígitos decimales
bool	1 bit	Almacena verdadero o falso
char	2 bytes	Almacena caracteres o letras rodeado de comillas simples
string	2 bytes por caracter	Almacena caracteres rodeado de comillas dobles

Números

Los tipos de números se dividen en dos grupos:

Los tipos de enteros almacenan números enteros, positivos o negativos (como 123 o -456), sin decimales. Los tipos válidos son `int` y `long`.

Los tipos de coma flotante representan números con una parte fraccionaria que contienen uno o más decimales. Los tipos válidos son `float` y `double`.

Tipos de números enteros

Tipos de números enteros

Int

El tipo de dato `int` puede almacenar números enteros desde -2147483648 hasta 2147483647.

```
int myNum = 100000;
Console.WriteLine(myNum);
```

Long

El tipo de dato `long` puede almacenar números enteros desde -9223372036854775808 hasta 9223372036854775807.
Esto se usa cuando `int` no es lo suficientemente grande para almacenar el valor. Tenga en cuenta que debe terminar el valor con una "L":

```
long myNum = 15000000000L;
Console.WriteLine(myNum);
```

Tipos de coma flotante

Tipos de coma flotante

Debe utilizar un tipo de coma flotante siempre que necesite un número con un decimal, como 9,99 o 3,14515.

Los tipos de datos `float` y `double` pueden almacenar números fraccionarios. Tenga en cuenta que debe finalizar el valor con una "F" para flotantes y una "D" para dobles:

```
float myNum = 5.75F;
Console.WriteLine(myNum);
```

```
double myNum = 19.99D;
Console.WriteLine(myNum);
```

Números científicos

Un número de coma flotante también puede ser un número científico con una "e" para indicar la potencia de 10:

```
float f1 = 35e3F;
double d1 = 12E4D;
Console.WriteLine(f1);
Console.WriteLine(d1);
```

Booleanos

Un tipo de datos booleano se declara con la palabra clave `bool` y solo puede tomar los valores `true` o `false`:

```
bool isCSharpFun = true;
bool isFishTasty = false;
Console.WriteLine(isCSharpFun);   // Outputs True
Console.WriteLine(isFishTasty);   // Outputs False
```

Los valores booleanos se utilizan principalmente para pruebas condicionales.

Caracteres

El tipo de dato `char` se utiliza para almacenar un solo carácter. El carácter debe estar entre comillas simples, como 'A' o 'c':

```
char myGrade = 'B';
Console.WriteLine(myGrade);
```

Strings

El tipo de dato `string` se utiliza para almacenar una secuencia de caracteres (texto). Los valores de cadena deben estar entre comillas dobles:

```
string greeting = "Hello World";
Console.WriteLine(greeting);
```

Conversión de tipo

Conversión de tipo

La conversión de tipos es cuando asigna un valor de un tipo de datos a otro tipo.

En C#, existen dos tipos de conversión:

- **Conversión implícita** (automáticamente): conversión de un tipo más pequeño a un tamaño de tipo más grande `char` -> `int` -> `long` -> `float` -> `double`
- **Conversión explícita** (manualmente): convertir un tipo más grande en un tipo de tamaño más pequeño
  `double` -> `float` -> `long` -> `int` -> `char`

Conversión implícita

La conversión implícita se realiza automáticamente al pasar un tipo de tamaño más pequeño a un tipo de tamaño más grande:

```
int myInt = 9;
double myDouble = myInt;       // Automatic casting: int to double

Console.WriteLine(myInt);      // Outputs 9
Console.WriteLine(myDouble);   // Outputs 9
```

Conversión explícita

La conversión explícita se debe realizar manualmente colocando el tipo entre paréntesis delante del valor:

```
double myDouble = 9.78;
int myInt = (int) myDouble;    // Manual casting: double to int

Console.WriteLine(myDouble);   // Outputs 9.78
Console.WriteLine(myInt);      // Outputs 9
```

Métodos de conversión de tipos

También es posible convertir tipos de datos explícitamente utilizando métodos integrados, como `Convert.ToBoolean`, `Convert.ToDouble`, `Convert.ToString`, `Convert.ToInt32` (int) y `Convert.ToInt64` (long):

```
int myInt = 10;
double myDouble = 5.25;
bool myBool = true;
```

```
Console.WriteLine(Convert.ToString(myInt));    // convert int to string
Console.WriteLine(Convert.ToDouble(myInt));    // convert int to double
Console.WriteLine(Convert.ToInt32(myDouble));  // convert double to int
Console.WriteLine(Convert.ToString(myBool));   // convert bool to string
```

Entrada de usuario

Entrada de usuario

Obtener información del usuario

Ya ha aprendido que `Console.WriteLine()` se utiliza para imprimir valores. Ahora lo usaremos `Console.ReadLine()` para obtener la entrada del usuario.

En el siguiente ejemplo, el usuario puede ingresar su nombre de usuario, que se almacena en la variable `userName`. Luego imprimimos el valor de `userName`:

```
// Type your username and press enter
Console.WriteLine("Enter username:");

// Create a string variable and get user input from the keyboard and store it in the
variable
string userName = Console.ReadLine();

// Print the value of the variable (userName), which will display the input value
Console.WriteLine("Username is: " + userName);
```

Entrada de usuario y números

El método `Console.ReadLine()` devuelve un string. Por lo tanto, no puede obtener información de otro tipo de datos, como `int`. El siguiente programa provocará un error:

```
Console.WriteLine("Enter your age:");
int age = Console.ReadLine();
Console.WriteLine("Your age is: " + age);
```

El mensaje de error será algo como esto:

```
Cannot implicitly convert type 'string' to 'int'
```

Como dice el mensaje de error, no se puede convertir implícitamente el tipo 'string' a 'int'.

Afortunadamente, acabas de aprender en el capítulo anterior (Conversión de tipos) que puedes convertir cualquier tipo explícitamente, utilizando uno de los métodos: `Convert.To`:

```
Console.WriteLine("Enter your age:");
int age = Convert.ToInt32(Console.ReadLine());
Console.WriteLine("Your age is: " + age);
```

Operadores

Operadores

Los operadores se utilizan para realizar operaciones con variables y valores.

En el siguiente ejemplo, utilizamos el operador `+` para sumar dos valores:

```
int x = 100 + 50;
```

Aunque el operador `+` se usa a menudo para sumar dos valores, como en el ejemplo anterior, también se puede usar para sumar una variable y un valor, o una variable y otra variable:

```
int sum1 = 100 + 50;         // 150 (100 + 50)
int sum2 = sum1 + 250;       // 400 (150 + 250)
int sum3 = sum2 + sum2;      // 800 (400 + 400)
```

Operadores aritméticos

Los operadores aritméticos se utilizan para realizar operaciones matemáticas comunes:

Tipo de dato	Tamaño	Descripcion	Ejemplo
+	Adicion	Añade juntos dos valores	x + y
-	Subtraccion	Sustrae un valor del otro	x - y
*	Multiplicacion	Multiplica dos valores	x * y
/	Divicion	Divide un valor por otro	x / y
%	Modulo	Devuelve el resto de la divicion	x % y
++	Incrementar	Incrementa el valor de una variable en 1	x++
--	Decrementar	Decrementa el valor de una variable en 1	x--

Operadores de asignación

Operadores de asignación

Los operadores de asignación se utilizan para asignar valores a variables.

En el siguiente ejemplo, utilizamos el operador de asignación `=` para asignar el valor 10 a una variable llamada x :

```
int x = 10;
```

El operador de asignación de suma `+=` agrega un valor a una variable:

```
int x = 10;
x += 5;
```

Una lista de todos los operadores de asignación:

Operador	Ejemplo	es como
`=`	`x = 5`	`x = 5`
`+=`	`x += 3`	`x = x + 3`
`-=`	`x -= 3`	`x = x - 3`
`*=`	`x *= 3`	`x = x * 3`
`/=`	`x /= 3`	`x = x / 3`
`%=`	`x %= 3`	`x = x % 3`
`&=`	`x &= 3`	`x = x & 3`
`|=`	`x |= 3`	`x = x | 3`
`^=`	`x ^= 3`	`x = x ^ 3`
`>>=`	`x >>= 3`	`x = x >> 3`
`<<=`	`x <<= 3`	`x = x << 3`

Operadores de comparación

Operadores de comparación

Los operadores de comparación se utilizan para comparar dos valores (o variables). Esto es importante en programación, porque nos ayuda a encontrar respuestas y tomar decisiones.

El valor de retorno de una comparación es `True` o `False`.

En el siguiente ejemplo, utilizamos el **operador mayor que** `>` para saber si 5 es mayor que 3:

```
int x = 5;
int y = 3;
Console.WriteLine(x > y); // returns True because 5 is greater than 3
```

Una lista de todos los operadores de comparación:

Operador	Nombre	Ejemplo
`==`	`igual a`	`x == y`
`!=`	`diferente a`	`x != y`
`>`	`mayor que`	`x > y`
`<`	`menor que`	`x < y`
`>=`	`mayor o igual a`	`x >= y`
`<=`	`menor o igual a`	`x <= y`

Operadores lógicos

Operadores lógicos

Al igual que con los operadores de comparación , también puede probar los valores `True` o `False` con operadores lógicos .

Los operadores lógicos se utilizan para determinar la lógica entre variables o valores:

Operador	Nombre	Descripcion	Ejemplo
`&&`	`logico and`	`Devuelve True si ambas afirmaciones son verdaderas`	`x < 5 && x < 10`
`||`	`logico or`	`Devuelve True si una de las afirmaciones es verdadera.`	`x < 5 || x < 4`

Operador	Nombre	Descripcion	Ejemplo
`!`	`logico not`	`Invierte el resultado, devuelve False si el resultado es true`	`!(x < 5 && x < 10)`

Matemáticas

Matemáticas

La clase Matematicas tienen muchos métodos que le permiten realizar tareas matemáticas con números.

Math.Max(x,y)

El método se puede utilizar para encontrar el valor más alto de x e y : `Math.Max(x,y)`

```
Math.Max(5, 10);
```

Math.Min(x,y)

El método se puede utilizar para encontrar el valor más bajo de x e y : `Math.Min(x,y)`

```
Math.Min(5, 10);
```

Math.Sqrt(x)

El método devuelve la raíz cuadrada de x : `Math.Sqrt(x)`

```
Math.Sqrt(64);
```

Math.Abs(x)

El método devuelve el valor absoluto (positivo) de x : `Math.Abs(x)`

```
Math.Abs(-4.7);
```

Math.Round()

`Math.Round()` redondea un número al número entero más cercano:

```
Math.Round(9.99);
```