



Sintaxis de



¿Qué es C #?

C # se pronuncia "C-Sharp".

Es un lenguaje de programación orientado a objetos creado por Microsoft que se ejecuta en .NET Framework.

C # tiene raíces de la familia C y el lenguaje está cerca de otros lenguajes populares como C ++ y Java .

La primera versión se lanzó en el año 2002. La última versión, C # 8 , se lanzó en septiembre de 2019.

Visual Studio 2019

Open recent

As you use Visual Studio, any projects, folders, or files that you open will show up here for quick access.

You can pin anything that you open frequently so that it's always at the top of the list.

Get started



Clone or check out code

Get code from an online repository like GitHub or Azure DevOps



Open a project or solution

Open a local Visual Studio project or .sln file



Open a local folder

Navigate and edit code within any folder



Create a new project

Choose a project template with code scaffolding to get started

[Continue without code →](#)

Create a new project

Recent project templates

A list of your recently accessed templates will be displayed here.

Search for templates (Alt+S)



All Languages

All Platforms

All Project Types



Blank Solution

Create an empty solution containing no projects

Other

Not finding what you're looking for?

[Install more tools and features](#)

Back

Next

Create a new project

Recent project templates

A list of your recently accessed templates will be displayed here.

All Languages

All Platforms

All Project Types



Console App (.NET Core)

New

A project for creating a command-line application that can run on .NET Core on Windows, Linux and MacOS.

C#

Linux

macOS

Windows

Console



Console App (.NET Core)

New

A project for creating a command-line application that can run on .NET Core on Windows, Linux and MacOS.

Visual Basic

Windows

Linux

macOS

Console



Class Library (.NET Standard)

New

A project for creating a class library that targets .NET Standard.

C#

Android

iOS

Linux

macOS

Windows

Library



Class Library (.NET Standard)

New

A project for creating a class library that targets .NET Standard.

Visual Basic

Android

iOS

Linux

macOS

Windows

Library



MSTest Test Project (.NET Core)

New

A project that contains MSTest unit tests that can run on .NET Core on Windows, Linux and MacOS.

C#

Linux

macOS

Windows

Test

Next

Configure your new project

Console App (.NET Core)

C#

Linux

macOS

Windows

Console

Project name


HelloWorld

Location

C:\Users\Username\source\repos

Solution

Create new solution

Solution name 

HelloWorld

☐ Place solution and project in the same directory

Back

Create

The screenshot displays the Visual Studio IDE interface. The top menu bar includes File, Edit, View, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, and Help. A search bar with the text 'Hell...orld' is visible. Below the menu bar is a toolbar with icons for file operations and a 'Debug' button. The main editor window shows the code for 'Program.cs' in the 'HelloWorld' project. The code is as follows:

```
1 using System;
2
3 namespace HelloWorld
4 {
5     class Program
6     {
7         static void Main(string[] args)
8         {
9             Console.WriteLine("Hello World!");
10        }
11    }
12 }
13
```

The right sidebar contains the 'Solution Explorer' pane, which shows the project structure: 'Solution 'HelloWorld' (1 of 1 project)' containing 'C# HelloWorld', which in turn contains 'Dependencies' and 'C# Program.cs'. The status bar at the bottom indicates '100 %' zoom, 'No issues found', and a 'Ready' state. There is also an 'Add to Source Control' button.

Program.cs

Línea 1: Significa que podemos usar clases del System espacio de nombres.

Línea 2: Línea en blanco. C # ignora los espacios en blanco.

Línea 7: Otra cosa que siempre aparece en un programa de C #, es el Main método. Se ejecutará cualquier código dentro de sus llaves. No es necesario que comprenda las palabras clave antes y después de Main.

Línea 9: Console es una clase del System espacio de nombres, que tiene un WriteLine() método que se usa para generar / imprimir texto. En nuestro ejemplo, generará ";Hola mundo!". Si omite la using System línea, tendrá que escribir System.Console.WriteLine() en el texto de impresión / salida.

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

Línea 3: namespace se utiliza para organizar su código y es un contenedor para clases y otros espacios de nombres.

Línea 4: Las llaves {} marcan el comienzo y el final de un bloque de código.

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

Tener en cuenta

Cada instrucción de C # termina con un punto y coma ;.
C # distingue entre mayúsculas y minúsculas: "MyClass" y "myclass" tienen un significado diferente.

WriteLine o Write

El método más común para generar algo en C # es WriteLine(), pero también puede usar Write().

La diferencia es que WriteLine() imprime la salida en una nueva línea cada vez, mientras que Write() imprime en la misma línea (tenga en cuenta que debe recordar agregar espacios cuando sea necesario, para una mejor legibilidad):

Ejemplo

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

Comentarios de C

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.

Los comentarios de una sola línea comienzan con dos barras diagonales (//).

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

Este ejemplo usa un comentario de una sola línea antes de una línea de código:

Ejemplo

```
// This is a comment
```

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

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

Comentarios de varias líneas de C

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

Cualquier texto entre `/*` y `*/` será ignorado por C #.

Este ejemplo utiliza un comentario de varias líneas (un bloque de comentarios) para explicar el código:

Ejemplo

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

Variables de C

Las variables son contenedores para almacenar valores de datos.

En C #, existen diferentes tipos de variables (definidas con diferentes palabras clave), por ejemplo:

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

Declarar (crear) variables

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

```
type variableName = value;
```

Ejemplo

```
string name = "John";  
Console.WriteLine(name);  
  
int myNum = 15;  
  
Console.WriteLine(myNum);
```

Ejemplo

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

Constantes

Sin embargo, puede agregar la const palabra clave si no desea que otros (o usted mismo) sobrescriban los valores existentes (esto declarará la variable como "constante", lo que significa inmutable y de solo lectura):

```
const int myNum = 15;
```

```
myNum = 20; // error
```

Otros tipos

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

Mostrar variables

El WriteLine() método se usa 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);
```

Declarar muchas variables

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

Identificadores de C

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

Estos nombres únicos se denominan identificadores .

Los identificadores pueden ser nombres cortos (como x e y) o nombres más descriptivos (edad, suma, volumen total).

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

Ejemplo

```
// Good
```

```
int minutesPerHour = 60;
```

```
// OK, but not so easy to understand what m  
// actually is
```

```
int m = 60;
```

Las reglas generales

Las reglas generales para construir nombres para variables (identificadores únicos) son:

- Los nombres pueden contener letras, dígitos y el carácter de subrayado (_)
- Los nombres deben comenzar con una letra
- 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 into double) no se pueden usar como nombres

Tipos de datos de C

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

Tipo más comunes

Un tipo de datos especifica el tamaño y el tipo de valores de variable. Es importante utilizar el tipo de datos correcto para la variable correspondiente; para evitar errores, para ahorrar tiempo y memoria, pero también hará que su código sea más fácil de mantener y leer.

Los tipos de datos más comunes son:

| | | |
|--------|-----------------------|---|
| int | 4 bytes | Stores whole numbers from -2,147,483,648 to 2,147,483,647 |
| long | 8 bytes | Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 |
| float | 4 bytes | Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits |
| double | 8 bytes | Stores fractional numbers. Sufficient for storing 15 decimal digits |
| bool | 1 bit | Stores true or false values |
| char | 2 bytes | Stores a single character/letter, surrounded by single quotes |
| string | 2 bytes per character | Stores a sequence of characters, surrounded by double quotes |

Los tipos de números se dividen en dos grupos:

Los tipos enteros almacenan números enteros, positivos o negativos (como 123 o -456), sin decimales. Los tipos válidos son int y long. El tipo que debe utilizar depende del valor numérico.

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 enteros - int

El int tipo de datos puede almacenar números enteros desde -2147483648 hasta 2147483647. En general, y en nuestro tutorial, el int tipo de datos es el tipo de datos preferido cuando creamos variables con un valor numérico.

```
int myNum = 100000;
```

```
Console.WriteLine(myNum);
```

Tipos de enteros - long

El long tipo de datos puede almacenar números enteros desde -9223372036854775808 hasta 9223372036854775807. Se utiliza 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 punto flotante - float

El float tipo de datos puede almacenar números fraccionarios de $3.4e - 038$ a $3.4e + 038$. Tenga en cuenta que debe terminar el valor con una "F":

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

Tipos de punto flotante - double

El double tipo de datos puede almacenar números fraccionarios de $1.7e - 308$ a $1.7e + 308$. Tenga en cuenta que puede terminar el valor con una "D" (aunque no es obligatorio):

```
double myNum = 19.99D;
```

```
Console.WriteLine(myNum);
```

¿Usar float o double?

La precisión de un valor de punto flotante indica cuántos dígitos puede tener el valor después del punto decimal. La precisión de float es de solo seis o siete dígitos decimales, mientras que las double variables tienen una precisión de aproximadamente 15 dígitos. Por lo tanto, es más seguro utilizarlo double para la mayoría de los cálculos.

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 bool palabra clave 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
```

Caracteres

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

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

Cadenas

El string tipo de datos 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);
```

Casting de tipo

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

En C #, hay dos tipos de conversión:

Conversión implícita (automáticamente): conversión de un tipo más pequeño a un tamaño de letra más grande

char-> int-> long-> float->double

Conversión explícita (manualmente): conversión de un tipo más grande en un tipo de tamaño más pequeño

double-> float-> long-> int->char

Casting implícito

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
```

Casting explícito

La conversión explícita se debe hacer 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
```

Caracteres

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

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

Métodos de conversión de tipos

También es posible convertir tipos de datos de forma explícita mediante el uso de una función de métodos, tales como `Convert.ToBoolean`, `Convert.ToDouble`, `Convert.ToString`, `Convert.ToInt32(int)` y `Convert.ToInt64(long)`:

Métodos de conversión de tipos

```
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

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

```
// 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 `Console.ReadLine()` método 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);
```

Entrada de usuario y números

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

Afortunadamente, para usted, acaba de aprender del capítulo anterior (Conversión de tipos) , que puede convertir cualquier tipo de forma explícita, utilizando uno de los Convert.To métodos:

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


Operadores aritméticos

| | | | |
|----|-------------------------------|--|----------|
| + | <u>Addition:</u> | Adds together two values | $x + y$ |
| - | <u>Subtraction:</u> | Subtracts one value from another | $x - y$ |
| * | <u>Multiplication:</u> | Multiplies two values | $x * y$ |
| / | <u>Division</u> | Divides one value by another | x / y |
| % | <u>Modulus</u> | Returns the division remainder | $x \% y$ |
| ++ | <u>Increment</u> | Increases the value of a variable by 1 | $x++$ |
| -- | <u>Decrement</u> | Decreases the value of a variable by 1 | $x--$ |

Operadores de asignación

| Operator | Example | Same As |
|----------|---------|------------|
| = | 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

| | | |
|----|--------------------------|----------|
| == | Equal to | $x == y$ |
| != | Not equal | $x != y$ |
| > | Greater than | $x > y$ |
| < | Less than | $x < y$ |
| >= | Greater than or equal to | $x >= y$ |
| <= | Less than or equal to | $x <= y$ |

Operadores lógicos

&& **Logical and** *Devuelve verdadero si ambas declaraciones son verdaderas*

$x < 5 \ \&\& \ x < 10$

|| **Logical or** *Devuelve verdadero si una de las declaraciones es verdadera*

$x < 5 \ || \ x < 4$

! **Logical not** *Invierte el resultado, devuelve falso si el resultado es verdadero*

$!(x < 5 \ \&\& \ x < 10)$

La clase C # Math tiene muchos métodos que le permiten realizar tareas matemáticas con números.

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

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

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

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

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

Cadenas

Las cadenas se utilizan para almacenar texto.

Una string variable contiene una colección de caracteres rodeados de comillas dobles:

```
string greeting = "Hello";
```

Una cadena en C # es en realidad un objeto, que contiene propiedades y métodos que pueden realizar ciertas operaciones en cadenas. Por ejemplo, la longitud de una cadena se puede encontrar con la Length propiedad:

```
string txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
```

```
Console.WriteLine("The length of the txt string is: "  
+ txt.Length);
```

Hay muchos métodos de cadena disponibles, por ejemplo ToUpper()y ToLower(), que devuelve una copia de la cadena convertida a mayúsculas o minúsculas:

```
string txt = "Hello World";  
Console.WriteLine(txt.ToUpper());    // Outputs  
// "HELLO WORLD"  
Console.WriteLine(txt.ToLower());    // Outputs  
// "hello world"
```


El + operador se puede utilizar entre cadenas para combinarlas. Esto se llama concatenación :

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

También puede usar el `string.Concat()` método para concatenar dos cadenas:

```
string firstName = "John ";  
string lastName = "Doe";  
string name = string.Concat(firstName, lastName);  
Console.WriteLine(name);
```

Otra opción de concatenación de cadenas es la interpolación de cadenas , que sustituye valores de variables en marcadores de posición en una cadena. Tenga en cuenta que no tiene que preocuparse por los espacios, como con la concatenación:

```
string firstName = "John";  
string lastName = "Doe";  
string name = $"My full name is: {firstName}  
{lastName}";  
Console.WriteLine(name);
```

Puede acceder a los caracteres de una cadena haciendo referencia a su número de índice entre corchetes [].

Este ejemplo imprime el primer carácter en myString :

```
string myString = "Hello";  
Console.WriteLine(myString[0]);  
// Outputs "H"
```

También puede encontrar la posición de índice de un carácter específico en una cadena, utilizando el `IndexOf()` método:

```
string myString = "Hello";  
Console.WriteLine(myString.IndexOf("e"));  
// Outputs "1"
```

Otro método útil es `Substring()`, que extrae los caracteres de una cadena, comenzando desde la posición / índice del carácter especificado, y devuelve una nueva cadena.

Este método se usa a menudo junto con `IndexOf()` para obtener la posición específica del personaje:

Ejemplo

```
// Full name
string name = "John Doe";

// Location of the letter D
int charPos = name.IndexOf("D");

// Get last name
string lastName = name.Substring(charPos);

// Print the result
Console.WriteLine(lastName);
```

| Escape character | Result | Description |
|------------------|--------|--------------|
| \' | ' | Single quote |
| \" | " | Double quote |
| \\ | \ | Backslash |

```
string txt = "We are the so-called \"Vikings\"  
from the north.";
```


Cadenas

Otros caracteres de escape útiles en C # son:

| <i>Code</i> | <i>Result</i> |
|--------------------|----------------------|
| <code>\n</code> | New Line |
| <code>\t</code> | Tab |
| <code>\b</code> | Backspace |

Valores booleanos

Un tipo booleano se declara con la bool palabra clave 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
```

Expresión booleana

Una expresión booleana es una expresión de C # que devuelve un valor booleano: True o False.

Puede usar un operador de comparación, como el operador mayor que (>) para averiguar si una expresión (o una variable) es verdadera:

```
int x = 10;  
int y = 9;  
Console.WriteLine(x > y); // returns True,  
// because 10 is higher than 9
```

Condiciones de C # y declaraciones If

C # admite las condiciones lógicas habituales de las matemáticas:

Menor que: $a < b$

Menor o igual a: $a \leq b$

Mayor que: $a > b$

Mayor o igual a: $a \geq b$

Igual a: $a == b$

No es igual a: $a \neq b$

Puede utilizar estas condiciones para realizar diferentes acciones para diferentes decisiones.

Condiciones de C # y declaraciones If

C # tiene las siguientes declaraciones condicionales:

Se usa **if** para especificar un bloque de código que se ejecutará, si una condición especificada es verdadera

Se usa **else** para especificar un bloque de código que se ejecutará, si la misma condición es falsa

Se usa **else if** para especificar una nueva condición para probar, si la primera condición es falsa

Use **switch** para especificar muchos bloques alternativos de código a ejecutar

La declaración if

Utilice la if declaración para especificar un bloque de código C # que se ejecutará si una condición es True.

```
if (condition)
{
    // block of code to be executed if the condition
    // is True
}
```

La declaración else

Utilice la else declaración para especificar un bloque de código que se ejecutará si la condición es False.

```
if (condition)
{
    // block of code to be executed if the condition is True
}
else
{
    // block of code to be executed if the condition is False
}
```

La declaración else if

```
if (condition1)
{
    // block of code to be executed if condition1 is True
}
else if (condition2)
{
    // block of code to be executed if the condition1 is false and
    // condition2 is True
}
else
{
    // block of code to be executed if the condition1 is false and
    // condition2 is False
}
```


Operador ternario

También hay una abreviatura if else, que se conoce como operador ternario porque consta de tres operandos. Se puede usar para reemplazar varias líneas de código con una sola línea. A menudo se usa para reemplazar declaraciones simples if else:

```
variable = (condition) ? expressionTrue : expressionFalse;  
  
int time = 20;  
string result = (time < 18) ? "Good day." : "Good evening.";   
Console.WriteLine(result);
```

Utilice la switch declaración para seleccionar uno de los muchos bloques de código que se ejecutarán.

```
switch(expression)
{
    case x:
        // code block
        break;
    case y:
        // code block
        break;
    default:
        // code block
        break;
}
```

Así es como funciona:

- La switch expresión se evalúa una vez
- El valor de la expresión se compara con los valores de cada case
- Si hay una coincidencia, se ejecuta el bloque de código asociado
- Las palabras clave break y default se describirán más adelante en este capítulo.
- Cuando C# llega a una break palabra clave, sale del bloque de interruptores.
- La default palabra clave es opcional y especifica algún código para ejecutar si no hay coincidencia de casos

Ciclo While

El while bucle recorre un bloque de código siempre que una condición especificada sea True:

```
while (condition)
{
    // code block to be executed
}
```

El bucle Do / While

El do/while bucle es una variante del while bucle. Este ciclo ejecutará el bloque de código una vez, antes de verificar si la condición es verdadera, luego repetirá el ciclo siempre que la condición sea verdadera.

```
do
{
    // code block to be executed
}
while (condition);
```

El ciclo For

Cuando sepa exactamente cuántas veces desea recorrer un bloque de código, use el for bucle en lugar de un while bucle:

```
for (statement 1; statement 2; statement 3)
{
    // code block to be executed
}
```

Así es como funciona:

- La instrucción 1 se ejecuta (una vez) antes de la ejecución del bloque de código.
- La declaración 2 define la condición para ejecutar el bloque de código.
- La instrucción 3 se ejecuta (cada vez) después de que se haya ejecutado el bloque de código.

El ciclo Foreach

También hay un foreach bucle, que se usa exclusivamente para recorrer elementos en una matriz:

```
foreach (type variableName in arrayName)
{
    // code block to be executed
}
```


Ejemplo

```
string[] cars = {"Volvo", "BMW",  
"Ford", "Mazda"};  
  
foreach (string i in cars)  
{  
    Console.WriteLine(i);  
}
```

Break

Ya ha visto la break declaración. Se utilizó para "saltar" de una switch declaración.

La break declaración también se puede utilizar para saltar de un bucle . Este ejemplo salta del ciclo cuando i es igual a 4:

```
for (int i = 0; i < 10; i++)  
{  
    if (i == 4)  
    {  
        break;  
    }  
    Console.WriteLine(i);  
}
```

Continue

La continue declaración rompe una iteración (en el ciclo), si ocurre una condición específica, y continúa con la siguiente iteración en el ciclo.

Este ejemplo omite el valor de 4:

```
for (int i = 0; i < 10; i++)  
{  
    if (i == 4)  
    {  
        continue;  
    }  
    Console.WriteLine(i);  
}
```

Arrays

Las matrices se utilizan para almacenar varios valores en una sola variable, en lugar de declarar variables independientes para cada valor.

Para declarar una matriz, defina el tipo de variable entre corchetes :

```
string[] cars;
```

```
string[] cars = {"Volvo", "BMW", "Ford",  
"Mazda"}; //Array literal
```

Acceder a los elementos de un Array

Puede acceder a un elemento de matriz haciendo referencia al número de índice.

```
string[] cars = {"Volvo", "BMW", "Ford", "Mazda"};  
Console.WriteLine(cars[0]);  
// Outputs Volvo  
  
cars[0] = "Opel";
```

Longitud de un vector

Para saber cuántos elementos tiene una matriz, use la Length propiedad:

```
string[] cars = {"Volvo", "BMW", "Ford",  
"Mazda"};  
Console.WriteLine(cars.Length);  
// Outputs 4
```

Bucle a través de un vector

Puede recorrer los elementos de la matriz con el for bucle y usar la propiedad Length para especificar cuántas veces debe ejecutarse el bucle.

```
string[] cars = {"Volvo", "BMW", "Ford", "Mazda"};  
for (int i = 0; i < cars.Length; i++)  
{  
    Console.WriteLine(cars[i]);  
}
```

Bucle a través de un vector

También hay un foreach bucle, que se usa exclusivamente para recorrer elementos en una matriz:

```
string[] cars = {"Volvo", "BMW", "Ford", "Mazda"};  
foreach (string i in cars)  
{  
    Console.WriteLine(i);  
}
```


Ordenar un vector

Hay muchos métodos de matriz disponibles, por ejemplo Sort(), que clasifican una matriz alfabéticamente o en orden ascendente:

```
// Sort a string
string[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
Array.Sort(cars);
foreach (string i in cars)
{
    Console.WriteLine(i);
}
```

Otras formas de crear Vectores

```
// Create an array of four elements, and add values later
string[] cars = new string[4];

// Create an array of four elements and add values right away
string[] cars = new string[4] {"Volvo", "BMW", "Ford", "Mazda"};

// Create an array of four elements without specifying the size
string[] cars = new string[] {"Volvo", "BMW", "Ford", "Mazda"};

// Create an array of four elements, omitting the new keyword,
// and without specifying the size
string[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
```

Métodos en C#

- Un método es un bloque de código que solo se ejecuta cuando se lo llama.
- Puede pasar datos, conocidos como parámetros, a un método.
- Los métodos se utilizan para realizar determinadas acciones y también se conocen como funciones.
- ¿Por qué utilizar métodos? Para reutilizar el código: defina el código una vez y utilícelo muchas veces.

Crear un método

Un método se define con el nombre del método, seguido de paréntesis (). C# proporciona algunos métodos predefinidos, con los que ya está familiarizado, como Main(), pero también puede crear sus propios métodos para realizar ciertas acciones:

```
class Program
{
    static void MyMethod()
    {
        // code to be executed
    }
}
```

Ejemplo explicado

- `MyMethod()` es el nombre del método
- `static` significa que el método pertenece a la clase `Program` y no a un objeto de la clase `Program`. Aprenderá más sobre objetos y cómo acceder a métodos a través de objetos más adelante.
- `void` significa que este método no tiene un valor de retorno. Aprenderá más sobre los valores devueltos más adelante.

Llamar a un método

Para llamar (ejecutar) un método, escriba el nombre del método seguido de dos paréntesis () y un punto y coma ;

En el siguiente ejemplo, MyMethod() se utiliza para imprimir un texto (la acción), cuando se llama:

```
static void MyMethod()  
{  
    Console.WriteLine("I just got executed!");  
}  
  
static void Main(string[] args)  
{  
    MyMethod();  
} // Outputs "I just got executed!"
```

Parámetros y argumentos

La información se puede pasar a los métodos como parámetro. Los parámetros actúan como variables dentro del método.

Se especifican después del nombre del método, entre paréntesis. Puede agregar tantos parámetros como desee, solo sepárelos con una coma.

El siguiente ejemplo tiene un método que toma un `fname string` llamado como parámetro. Cuando se llama al método, pasamos un nombre, que se usa dentro del método para imprimir el nombre completo:

Ejemplo

```
static void MyMethod(string fname)
{
    Console.WriteLine(fname + " Refsnes");
}

static void Main(string[] args)
{
    MyMethod("Liam");
    MyMethod("Jenny");
    MyMethod("Anja");
}

// Liam Refsnes
// Jenny Refsnes
// Anja Refsnes
```


Valor de parámetro predeterminado

```
static void MyMethod(string country = "Norway")
{
    Console.WriteLine(country);
}

static void Main(string[] args)
{
    MyMethod("Sweden");
    MyMethod("India");
    MyMethod();
    MyMethod("USA");
}

// Sweden
// India
// Norway
// USA
```

Múltiples parámetros

```
static void MyMethod(string fname, int age)
{
    Console.WriteLine(fname + " is " + age);
}

static void Main(string[] args)
{
    MyMethod("Liam", 5);
    MyMethod("Jenny", 8);
    MyMethod("Anja", 31);
}

// Liam is 5
// Jenny is 8
// Anja is 31
```

Valores devueltos

La void palabra clave, utilizada en los ejemplos anteriores, indica que el método no debe devolver un valor. Si desea que el método devuelva un valor, puede usar un tipo de datos primitivo (como int o double) en lugar de void, y usar la return palabra clave dentro del método:

Valores devueltos

```
static int MyMethod(int x)
{
    return 5 + x;
}

static void Main(string[] args)
{
    Console.WriteLine(MyMethod(3));
}

// Outputs 8 (5 + 3)
```

Argumentos nombrados

```
static void MyMethod(string child1, string child2, string
child3)
{
    Console.WriteLine("The youngest child is: " + child3);
}

static void Main(string[] args)
{
    MyMethod(child3: "John", child1: "Liam", child2: "Liam");
}

// The youngest child is: John
```

Sobrecarga de métodos

Con la sobrecarga de métodos, varios métodos pueden tener el mismo nombre con diferentes parámetros:

```
int MyMethod(int x)
float MyMethod(float x)
double MyMethod(double x, double y)
```

Ejemplo sin sobrecarga

```
static int PlusMethodInt(int x, int y)
{
    return x + y;
}

static double PlusMethodDouble(double x, double y)
{
    return x + y;
}

static void Main(string[] args)
{
    int myNum1 = PlusMethodInt(8, 5);
    double myNum2 = PlusMethodDouble(4.3, 6.26);
    Console.WriteLine("Int: " + myNum1);
    Console.WriteLine("Double: " + myNum2);
}
```

Ejemplo con sobrecarga

```
static int PlusMethod(int x, int y)
{
    return x + y;
}

static double PlusMethod(double x, double y)
{
    return x + y;
}

static void Main(string[] args)
{
    int myNum1 = PlusMethod(8, 5);
    double myNum2 = PlusMethod(4.3, 6.26);
    Console.WriteLine("Int: " + myNum1);
    Console.WriteLine("Double: " + myNum2);
}
```


Excepciones de C # - Try..Catch

Al ejecutar código C #, pueden ocurrir diferentes errores: errores de codificación cometidos por el programador, errores debido a una entrada incorrecta u otros imprevistos.

Cuando ocurre un error, C # normalmente se detendrá y generará un mensaje de error. El término técnico para esto es: C # lanzará una excepción (lanzará un error).

Try - Catch

La **try** declaración le permite definir un bloque de código para ser probado en busca de errores mientras se ejecuta.

La **catch** declaración le permite definir un bloque de código a ejecutar, si ocurre un error en el bloque try.

Las palabras clave try y catch vienen en pares.

Si ocurre un error, podemos usarlo try...catch para detectar el error y ejecutar algún código para manejarlo.

```
try
{
    // Block of code to try
}
catch (Exception e)
{
    // Block of code to handle errors
}
```

Ejemplo

```
try
{
    int[] myNumbers = {1, 2, 3};
    Console.WriteLine(myNumbers[10]);
}
catch (Exception e)
{
    Console.WriteLine(e.Message);
}
```

```
try
{
    int[] myNumbers = {1, 2, 3};
    Console.WriteLine(myNumbers[10]);
}
catch (Exception e)
{
    Console.WriteLine("Something went wrong.");
}
finally
{
    Console.WriteLine("The 'try catch' is finished.");
}
```

La palabra clave throw

La throw declaración le permite crear un error personalizado.

La throw declaración se usa junto con una clase de excepción .

Hay muchas clases de excepciones disponibles en C #:

ArithmeticException, FileNotFoundException,

IndexOutOfRangeException, TimeoutException, etc:

Ejemplo

```
static void checkAge(int age)
{
    if (age < 18)
    {
        throw new ArithmeticException("Access denied - You must be at least 18 years old.");
    }
    else
    {
        Console.WriteLine("Access granted - You are old enough!");
    }
}

static void Main(string[] args)
{
    checkAge(15);
}
```

Debugger

ConsoleApp2 - Microsoft Visual Studio

Archivo Editar Ver NCrunch Proyecto Compilar Depurar Equipo Herramientas Pruebas ReSharper Analizar Ventana Ayuda

Debug Any CPU ConsoleApp2

Program.cs

```
1 using System;
2
3 namespace ConsoleApp2
4 {
5     class Program
6     {
7         static void Main(string[] args)
8         {
9             Console.WriteLine("Hello World!");
10        }
11    }
12 }
~
```

56 %

Explorador de soluciones

Buscar en Explorador de soluciones (Ctrl+)

Solución "ConsoleApp2" (1 proyecto)

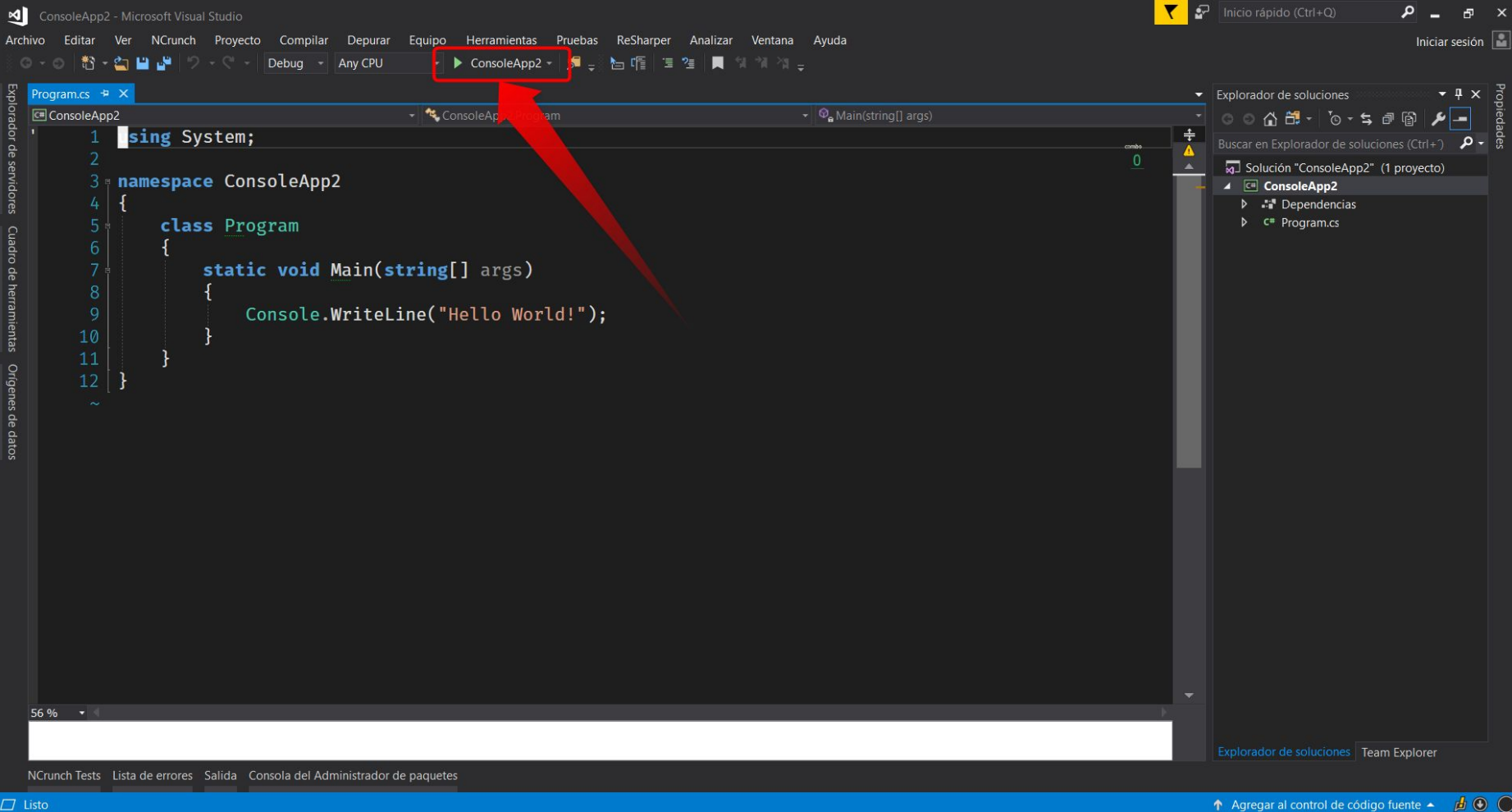
- ConsoleApp2
 - Dependencias
 - Program.cs

Explorador de soluciones Team Explorer

NCrunch Tests Lista de errores Salida Consola del Administrador de paquetes

Listo

Agregar al control de código fuente



ConsoleApp2 - Microsoft Visual Studio

Archivo Editar Ver NCrunch Proyecto Compilar Depurar Equipo Herramientas Pruebas ReSharper Analizar Ventana Ayuda

Debug Any CPU ConsoleApp2

Program.cs

ConsoleApp2 ConsoleApp2.Program Main(string[] args)

```
1 using System;
2
3 namespace ConsoleApp2
4 {
5     class Program
6     {
7         static void Main(string[] args)
8         {
9             Console.WriteLine("Hello World!");
10
11             Console.ReadLine();
12         }
13     }
14 }
```

56 %

Explorador de soluciones

Buscar en Explorador de soluciones (Ctrl+)

Solución "ConsoleApp2" (1 proyecto)

- ConsoleApp2
 - Dependencias
 - Program.cs

Explorador de soluciones Team Explorer

NCrunch Tests Lista de errores Salida Consola del Administrador de paquetes

Listo

Agregar al control de código fuente

ConsoleApp2 (Depuración) - Microsoft Visual Studio

Archivo Editar Ver NCrunch Proyecto Compilar Depurar Equipo Herramientas Pruebas ReSharper Analizar Ventana Ayuda

Proceso: [564] dotnet.exe Eventos del ciclo de vida Subproceso: [2288] Subproceso principal Marco de pila: ConsoleApp2.Program.Main

Program.cs

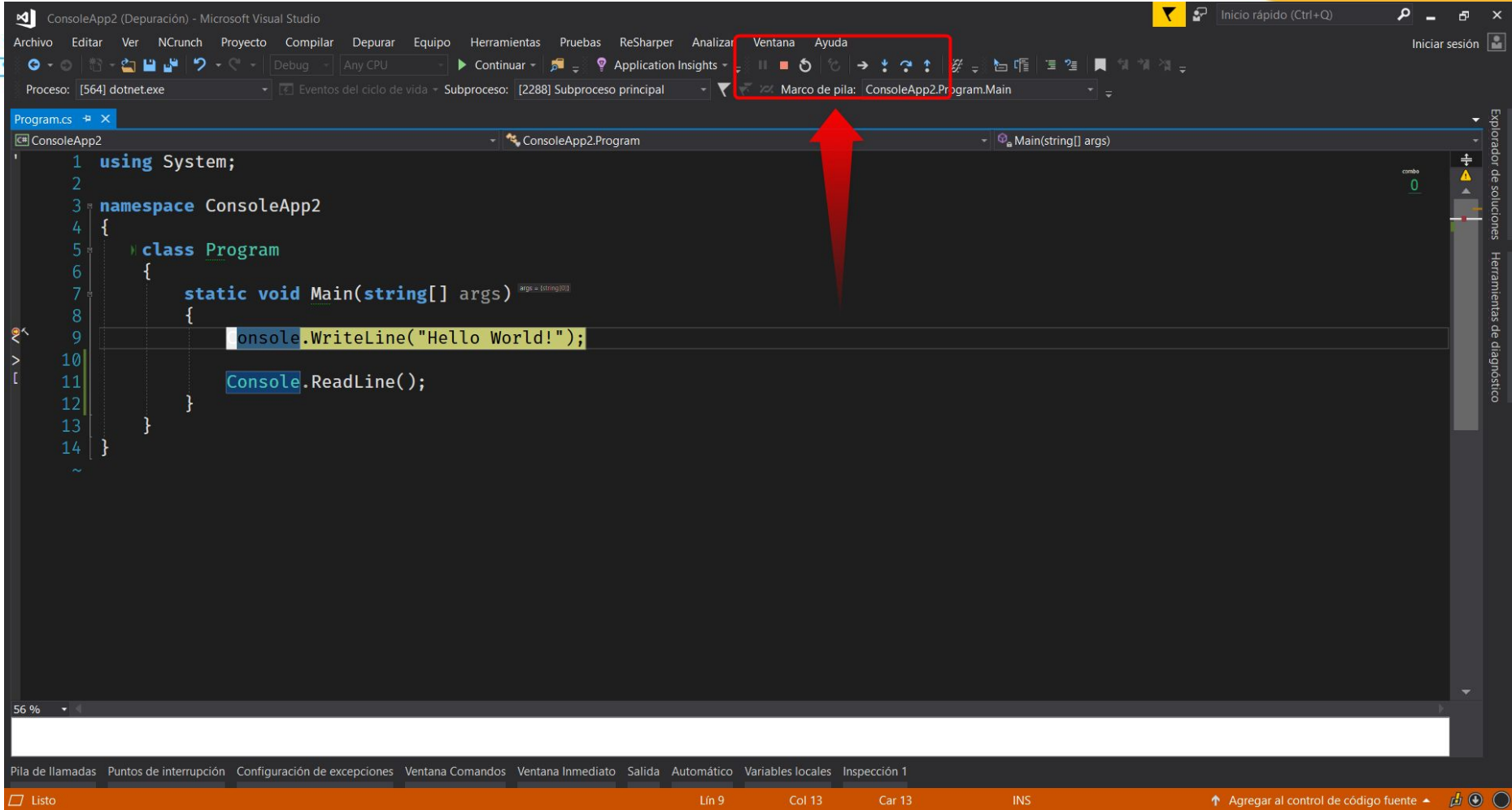
ConsoleApp2 ConsoleApp2.Program Main(string[] args)

```
1 using System;
2
3 namespace ConsoleApp2
4 {
5     class Program
6     {
7         static void Main(string[] args)
8         {
9             Console.WriteLine("Hello World!");
10
11             Console.ReadLine();
12         }
13     }
14 }
```

56 %

Pila de llamadas Puntos de interrupción Configuración de excepciones Ventana Comandos Ventana Inmediato Salida Automático Variables locales Inspección 1

Listo Lín 9 Col 13 Car 13 INS Agregar al control de código fuente

A screenshot of the Microsoft Visual Studio IDE. The main window displays a C# file named Program.cs. The code defines a namespace ConsoleApp2, a class Program, and a static void Main method. The Main method contains two lines: Console.WriteLine("Hello World!"); and Console.ReadLine(). The status bar at the bottom indicates the current position is Line 9, Column 13, Character 13. On the right side, there is a vertical toolbar with icons for 'Explorador de soluciones' (Solution Explorer), 'Herramientas de diagnóstico' (Diagnostic Tools), and a 'combinar' (Combine) button with the number 0. A red arrow points from the 'Marco de pila' (Stack) window, which is highlighted in the top toolbar, towards the Main method in the code editor.

Ejemplos

ConsoleApp2 - Microsoft Visual Studio

Inicio rápido (Ctrl+Q) Iniciar sesión

Archivo Editar Ver NCrunch Proyecto Compilar Depurar Equipo Herramientas Pruebas ReSharper Analizar Ventana Ayuda

Debug Any CPU ConsoleApp2

Program.cs ConsoleApp2 ConsoleApplication1.Program Main(string[] args)

```
1 using System;
2
3 namespace ConsoleApplication1
4 {
5     class Program
6     {
7         static void Main(string[] args)
8         {
9             int NUM1, NUM2, RESUL;
10            string linea;
11            Console.Write("PRIMER NÚMERO :");
12            linea = Console.ReadLine();
13            NUM1 = int.Parse(linea);
14            Console.Write("SEGUNDO NÚMERO :");
15            linea = Console.ReadLine();
16            NUM2 = int.Parse(linea);
17            Console.WriteLine();
18            RESUL = NUM1 + NUM2;
19            Console.WriteLine("LA SUMA ES {0}: ", RESUL);
20            RESUL = NUM1 - NUM2;
21            Console.WriteLine("LA RESTA ES: {0} - {1} = {2} ", NUM1, NUM2, RESUL);
22            RESUL = NUM1 * NUM2;
23            Console.WriteLine("LA MULTIPLICACIÓN ES: " + RESUL);
24            RESUL = NUM1 / NUM2;
25            Console.WriteLine("LA DIVISIÓN ES: " + RESUL);
26            RESUL = NUM1 % NUM2;
27            Console.WriteLine("EL RESIDUO ES: " + RESUL);
28            Console.Write("Pulse una Tecla:");
29            Console.ReadLine();
30        }
31    }
32 }
```

46 %

Explorador de soluciones

Buscar en Explorador de soluciones (Ctrl+)

Solución "ConsoleApp2" (1 proyecto)

- ConsoleApp2
 - Dependencias
 - Program.cs

Explorador de soluciones Team Explorer

NCrunch Tests Lista de errores Salida Consola del Administrador de paquetes

Listo Lín 1 Col 14 Car 14 INS

Agregar al control de código fuente

ConsoleApp2 - Microsoft Visual Studio

Archivo Editar Ver NCrunch Proyecto Compilar Depurar Equipo Herramientas Pruebas ReSharper Analizar Ventana Ayuda

Debug Any CPU ConsoleApp2

Program.cs

ConsoleApp2 ConsoleApplication1.Program Main(string[] args)

```
1 using System;
2
3 namespace ConsoleApplication1
4 {
5     class Program
6     {
7         static void Main(string[] args)
8         {
9             int NUM1;
10            string linea;
11            long RESUL;
12            Console.Write("DIGITE UN NÚMERO :");
13            linea = Console.ReadLine();
14            NUM1 = int.Parse(linea);
15            RESUL = Math.Abs(NUM1);
16            Console.WriteLine("VALOR ABSOLUTO : " + RESUL);
17            Console.WriteLine("POTENCIA : " + Math.Pow(NUM1, 3));
18            Console.WriteLine("RAIZ CUADRADA : " + Math.Sqrt(NUM1));
19            Console.WriteLine("SENO : " + Math.Sin(NUM1 * Math.PI / 180));
20            Console.WriteLine("COSENO : " + Math.Cos(NUM1 * Math.PI / 180));
21            Console.WriteLine("NÚMERO MÁXIMO : " + Math.Max(NUM1, 50));
22            Console.WriteLine("NÚMERO MÍNIMO : " + Math.Min(NUM1, 50));
23            Console.WriteLine("PARTE ENTERA : " + Math.Truncate(18.78));
24            Console.WriteLine("REDONDEO : " + Math.Round(18.78));
25            Console.Write("Pulse una Tecla:"); Console.ReadLine();
26        }
27    }
28 }
```

51 %

-- INSERT --

NCrunch Tests Lista de errores Salida Consola del Administrador de paquetes

Inicio rápido (Ctrl+Q)

Explorador de soluciones

Buscar en Explorador de soluciones (Ctrl+)

Solución "ConsoleApp2" (1 proyecto)

- ConsoleApp2
 - Dependencias
 - Program.cs

Explorador de soluciones Team Explorer

Listo

Agregar al control de código fuente

ConsoleApp2 - Microsoft Visual Studio

Archivo Editar Ver NCrunch Proyecto Compilar Depurar Equipo Herramientas Pruebas ReSharper Analizar Ventana Ayuda

Debug Any CPU ConsoleApp2

Program.cs

ConsoleApp2 ConsoleApplication1.Program Main(string[] args)

```
1 using System;
2
3 namespace ConsoleApplication1
4 {
5     class Program
6     {
7         static void Main(string[] args)
8         {
9             double BASE, ALTURA, RESUL;
10            string linea;
11            Console.Write("DIGITE LA BASE :"); linea = Console.ReadLine();
12            BASE = double.Parse (linea);
13            Console.Write("DIGITE LA ALTURA:"); linea = Console.ReadLine();
14            ALTURA= double.Parse (linea);
15            RESUL = (BASE * ALTURA) / 2;
16            Console.WriteLine("AREA TRIANGULO : " + String.Format("{0:###.00}", RESUL));
17            Console.WriteLine("AREA TRIANGULO : " + String.Format("{0:c}", RESUL));
18            Console.WriteLine("AREA TRIANGULO : " + String.Format("{0:f}", RESUL));
19            Console.WriteLine("AREA TRIANGULO : " + String.Format("{0:g}", RESUL));
20            Console.WriteLine();
21            Console.WriteLine("HOY ES : " + String.Format("Hoy es {0:F}", DateTime.Now));
22            Console.WriteLine("HOY ES : " + String.Format("Hoy es {0:dddd}{0:dd/MM/yyyy}",
23                DateTime.Now));
24            Console.Write("Pulse una Tecla:"); Console.ReadLine();
25        }
26    }
27 }
28
```

51 %

26

Explorador de soluciones

Buscar en Explorador de soluciones (Ctrl+)

Solución "ConsoleApp2" (1 proyecto)

- ConsoleApp2
 - Dependencias
 - Program.cs

Explorador de soluciones Team Explorer

NCrunch Tests Lista de errores Salida Consola del Administrador de paquetes

Listo

Agregar al control de código fuente

Ejercicios

Invertir número de dos cifras

Dado un número ingresado por teclado de 2 cifras se pide un programa que muestre el resultado de invertir dicho número.

Por ejemplo, si se ingresa 12 debe mostrar el resultado de 21.

ConsoleApp2 - Microsoft Visual Studio

Inicio rápido (Ctrl+Q) Iniciar sesión

Archivo Editar Ver NCrunch Proyecto Compilar Depurar Equipo Herramientas Pruebas ReSharper Analizar Ventana Ayuda

Debug Any CPU ConsoleApp2

Program.cs

ConsoleApp2 ConsoleApplication1.Program Main(string[] args)

```
1 using System;
2
3 namespace ConsoleApplication1
4 {
5     class Program
6     {
7         static void Main(string[] args)
8         {
9             int NUM, AUX, DEC, UNI;
10            string linea;
11            Console.WriteLine("INGRESE NÚMERO DE DOS CIFRAS :");
12            linea = Console.ReadLine();
13            NUM = int.Parse(linea);
14            DEC = NUM/10;
15            UNI = NUM % 10;
16            AUX = (UNI * 10) + DEC;
17            Console.WriteLine("NÚMERO INVERTIDO ES: " + AUX);
18            Console.WriteLine("Pulse una Tecla:"); Console.ReadLine();
19        }
20    }
21 }
```

56 %

Explorador de soluciones

Buscar en Explorador de soluciones (Ctrl+)

Solución "ConsoleApp2" (1 proyecto)

- ConsoleApp2
 - Dependencias
 - Program.cs

Explorador de soluciones Team Explorer

NCrunch Tests Lista de errores Salida Consola del Administrador de paquetes

Listo Lín 2 Col 1 Car 1 INS Agregar al control de código fuente

Operación

El usuario debe ingresar dos números y el programa mostrará el resultado de la operación $(a+b)*(a-b)$

ConsoleApp2 - Microsoft Visual Studio

Inicio rápido (Ctrl+Q) Iniciar sesión

Archivo Editar Ver NCrunch Proyecto Compilar Depurar Equipo Herramientas Pruebas ReSharper Analizar Ventana Ayuda

Debug Any CPU ConsoleApp2

Program.cs* x

ConsoleApp2 Ejercicio_propuesto_1.Program Main(string[] args)

```
1 using System;
2
3 namespace Ejercicio_propuesto_1
4 {
5     class Program
6     {
7         static void Main(string[] args)
8         {
9             int NUM1, NUM2;
10            double RESUL;
11            string linea;
12            Console.WriteLine("NÚMERO 1 :"); linea = Console.ReadLine();
13            NUM1 = int.Parse(linea);
14            Console.WriteLine("NÚMERO 2 :"); linea = Console.ReadLine();
15            NUM2 = int.Parse(linea);
16            RESUL = (NUM1 + NUM2) * (NUM1 - NUM2);
17            Console.WriteLine();
18            Console.WriteLine("El resultado es : " +RESUL );
19            Console.ReadLine();
20        }
21    }
22 }
```

62 %

4 lines deleted

NCrunch Tests Lista de errores Salida Consola del Administrador de paquetes

Explorador de soluciones

Buscar en Explorador de soluciones (Ctrl+)

Solución "ConsoleApp2" (1 proyecto)

- ConsoleApp2
 - Dependencias
 - Program.cs

Explorador de soluciones Team Explorer

Lín 1 Col 14 Car 14 INS

Agregar al control de código fuente

Número o vocal o consonante

Crear un programa que lea una letra tecleada por el usuario y diga si se trata de una vocal, una cifra numérica o una consonante (pista: habrá que usar un dato de tipo "char").

ConsoleApp2 - Microsoft Visual Studio

Inicio rápido (Ctrl+Q) Iniciar sesión

Archivo Editar Ver NCrunch Proyecto Compilar Depurar Equipo Herramientas Pruebas ReSharper Analizar Ventana Ayuda

Debug Any CPU ConsoleApp2

Program.cs* X

ConsoleApp2 Ejercicio_Propuesto_3.Program Main(string[] args)

```
2
3 namespace Ejercicio_Propuesto_3
4 {
5     class Program
6     {
7         static void Main(string[] args)
8         {
9             char TECLA;
10            string linea;
11            Console.Write("DIGITE UNA SOLA TECLA");
12            Console.WriteLine();
13            linea = Console.ReadLine();
14            TECLA = char.Parse(linea);
15            switch (TECLA)
16            {
17                case '1':
18                case '2':
19                case '3':
20                case '4':
21                case '5':
22                case '6':
23                case '7':
24                case '8':
25                case '9' : ; Console.WriteLine("ES UNA CIFRA NUMERICA");
26                break;
27                case 'a':
```

56 %

-- INSERT --

Explorador de soluciones

Buscar en Explorador de soluciones (Ctrl+)

Solución "ConsoleApp2" (1 proyecto)

- ConsoleApp2
 - Dependencias
 - Program.cs

Explorador de soluciones Team Explorer

NCrunch Tests Lista de errores Salida Consola del Administrador de paquetes

Listo

Agregar al control de código fuente

ConsoleApp2 - Microsoft Visual Studio

Inicio rápido (Ctrl+Q) Iniciar sesión

Archivo Editar Ver NCrunch Proyecto Compilar Depurar Equipo Herramientas Pruebas ReSharper Analizar Ventana Ayuda

Debug Any CPU ConsoleApp2

Program.cs* Ejercicio_Propuesto_3.Program Main(string[] args)

```
24     case '8':
25     case '9': ; Console.WriteLine("ES UNA CIFRA NUMERICA");
26         break;
27     case 'a':
28     case 'e':
29     case 'i':
30     case 'o':
31     case 'u': ; Console.WriteLine("ES UNA VOCAL");
32         break;
33     default:
34         Console.WriteLine("ES UNA CONSONANTE");
35         break;
36     }
37     Console.Write("Pulse una Tecla:"); Console.ReadLine();
38 }
39 }
40 }
```

56 %

-- INSERT --

NCrunch Tests Lista de errores Salida Consola del Administrador de paquetes

Explorador de soluciones

Buscar en Explorador de soluciones (Ctrl+)

Solución "ConsoleApp2" (1 proyecto)

- ConsoleApp2
 - Dependencias
 - Program.cs

Team Explorer

Explorador de soluciones

Team Explorer

Agregar al control de código fuente

Lín 1 Col 14 Car 14 INS

Letras

Crear un programa que muestre las letras de la Z (mayúscula) a la A (mayúscula, descendiendo).

ConsoleApp2 - Microsoft Visual Studio

Archivo Editar Ver NCrunch Proyecto Compilar Depurar Equipo Herramientas Pruebas ReSharper Analizar Ventana Ayuda

Debug Any CPU ConsoleApp2

Program.cs* x

ConsoleApp2 EJERCICIO_PROPUESTO_4.Program Main(string[] args)

```
1 using System;
2
3 namespace EJERCICIO_PROPUESTO_4
4 {
5     class Program
6     {
7         static void Main(string[] args)
8         {
9             int i;
10            Console.Write("ABECEDARIO DESCENDENTE");
11            Console.WriteLine();
12            for (i = 90; i ≥ 65 ; i--)
13            {
14                Console.WriteLine("LETRA: " + Convert.ToString((char)i));
15            }
16            Console.Write("Pulse una Tecla:"); Console.ReadLine();
17        }
18    }
19 }
```

68 %

-- INSERT --

NCrunch Tests Lista de errores Salida Consola del Administrador de paquetes

Explorador de soluciones

Buscar en Explorador de soluciones (Ctrl+)

Solución "ConsoleApp2" (1 proyecto)

- ConsoleApp2
 - Dependencias
 - Program.cs

Explorador de soluciones Team Explorer

Lín 1 Col 14 Car 14 INS

↑ Agregar al control de código fuente

Cuántas cifras

Crear un programa calcule cuántas cifras tiene un número entero positivo (pista: se puede hacer dividiendo varias veces entre 10).

ConsoleApp2 - Microsoft Visual Studio

Archivo Editar Ver NCrunch Proyecto Compilar Depurar Equipo Herramientas Pruebas ReSharper Analizar Ventana Ayuda

Debug Any CPU ConsoleApp2

Program.cs*

ConsoleApp2 Ejercicio_Propuesto_5.Program Main(string[] args)

```
1 using System;
2
3 namespace Ejercicio_Propuesto_5
4 {
5     class Program
6     {
7         static void Main(string[] args)
8         {
9             int NUM, I, Z, cont=0;
10            string linea;
11            Console.Write("DIGITE NÚMERO:"); linea = Console.ReadLine();
12            NUM = int.Parse(linea);
13            Z = linea.Length;
14            I = Z;
15            while ((I >= 1))
16            {
17                I--;
18                cont++;
19            }
20            Console.WriteLine();
21            Console.WriteLine("El numero ingresado tiene " + cont + " digitos");
22            Console.Write("Pulse una Tecla:"); Console.ReadLine();
23
24        }
25    }
26 }
```

56 %

-- VISUAL --

1

Explorador de soluciones

Buscar en Explorador de soluciones (Ctrl+)

Solución "ConsoleApp2" (1 proyecto)

- ConsoleApp2
 - Dependencias
 - Program.cs

Explorador de soluciones Team Explorer

NCrunch Tests Lista de errores Salida Consola del Administrador de paquetes

Lín 1 Col 13 Car 13 INS

Agregar al control de código fuente

Suma

Crear un programa que pida números positivos al usuario, y vaya calculando la suma de todos ellos (terminará cuando se teclea un número negativo o cero).

ConsoleApp2 - Microsoft Visual Studio

Inicio rápido (Ctrl+Q) Iniciar sesión

Archivo Editar Ver NCrunch Proyecto Compilar Depurar Equipo Herramientas Pruebas ReSharper Analizar Ventana Ayuda

Debug Any CPU ConsoleApp2

Program.cs* Ejercicio_Propuesto_6.Program Main(string[] args)

```
1 using System;
2
3 namespace Ejercicio_Propuesto_6
4 {
5     class Program
6     {
7         static void Main(string[] args)
8         {
9             int NUM, RES, TOT=0 ;
10            string linea;
11            do{
12                Console.Write("INGRESE UN NUMERO :");
13                linea = Console.ReadLine();
14                NUM = int.Parse(linea);
15                RES = NUM % 2;
16                if (RES == 0 && NUM != 0)
17                    TOT = TOT + NUM;
18                else
19                {
20                }
21            }while(RES == 0 && NUM !=0);
22
23            Console.WriteLine ("La suma total es: " +TOT);
24            Console.Write("Pulse una Tecla:");
25            Console.ReadLine();
26        }
27    }
28 }
29
```

51 %

-- INSERT --

NCrunch Tests Lista de errores Salida Consola del Administrador de paquetes

Explorador de soluciones

Buscar en Explorador de soluciones (Ctrl+)

Solución "ConsoleApp2" (1 proyecto)

- ConsoleApp2
 - Dependencias
 - Program.cs

Explorador de soluciones Team Explorer

Lín 1 Col 14 Car 14 INS

Agregar al control de código fuente

Operarios

Confeccionar un programa que permita cargar los nombres de 5 operarios y sus sueldos respectivos. Mostrar el sueldo mayor y el nombre del operario.

ConsoleApp2 - Microsoft Visual Studio

Inicio rápido (Ctrl+Q) Iniciar sesión

Archivo Editar Ver NCrunch Proyecto Compilar Depurar Equipo Herramientas Pruebas ReSharper Analizar Ventana Ayuda

Debug Any CPU ConsoleApp2

Program.cs

ConsoleApp2 PruebaVector11.PruebaVector11 nombres

```
1 using System;
2
3 namespace PruebaVector11
4 {
5     class PruebaVector11
6     {
7         private string[] nombres;
8         private float[] sueldos;
9         public void Cargar()
10        {
11            nombres = new string[5];
12            sueldos = new float[5];
13            for (int f = 0; f < nombres.Length; f++)
14            {
15                Console.WriteLine("Ingrese el nombre del empleado:");
16                nombres[f] = Console.ReadLine();
17                Console.WriteLine("Ingrese el sueldo:");
18                string linea;
19                linea = Console.ReadLine();
20                sueldos[f] = float.Parse(linea);
21            }
22        }
23        public void MayorSueldo()
24        {
25            float mayor;
26            int pos;
27            mayor = sueldos[0];
28            pos = 0;
29            for (int f = 1; f < nombres.Length; f++)
```

51 %

Explorador de soluciones

Buscar en Explorador de soluciones (Ctrl+)

Solución "ConsoleApp2" (1 proyecto)

- ConsoleApp2
 - Dependencias
 - Program.cs

Explorador de soluciones Team Explorer

NCrunch Tests Lista de errores Salida Consola del Administrador de paquetes

Listo Lín 2 Col 1 Car 1 INS

Agregar al control de código fuente

ConsoleApp2 - Microsoft Visual Studio

Inicio rápido (Ctrl+Q) Iniciar sesión

Archivo Editar Ver NCrunch Proyecto Compilar Depurar Equipo Herramientas Pruebas ReSharper Analizar Ventana Ayuda

Debug Any CPU ConsoleApp2

Program.cs* x

ConsoleApp2 PruebaVector11.PruebaVector11 nombres

```
21 }
22 }
23 public void MayorSueldo()
24 {
25     float mayor;
26     int pos;
27     mayor = sueldos[0];
28     pos = 0;
29     for (int f = 1; f < nombres.Length; f++)
30     {
31         if (sueldos[f] > mayor)
32         {
33             mayor = sueldos[f];
34             pos = f;
35         }
36     }
37     Console.WriteLine("El empleado con sueldo mayor es " + nombres[pos]);
38     Console.WriteLine("Tiene un sueldo:" + mayor);
39     Console.ReadKey();
40 }
41 static void Main(string[] args)
42 {
43     PruebaVector11 pv = new PruebaVector11();
44     pv.Cargar();
45     pv.MayorSueldo();
46 }
47 }
48 }
49 }
```

(parameter) string value

51 %

Explorador de soluciones

Buscar en Explorador de soluciones (Ctrl+)

Solución "ConsoleApp2" (1 proyecto)

- ConsoleApp2
 - Dependencias
 - Program.cs

Explorador de soluciones Team Explorer

NCrunch Tests Lista de errores Salida Consola del Administrador de paquetes

Listo Lín 2 Col 1 Car 1 INS

Agregar al control de código fuente

Preguntas?

