



Desarrollamos
{ talento tech }



GeeksHubs
academy_



CSharp

Introducción al lenguaje.



Historia de CSharp





C# es un lenguaje orientado a objetos, utilizando un sistema de código gestionado y creado en el año 1999 y su creador fue [Andres Hejlsberg](#), posteriormente viendo la evolución del lenguaje y su potencial fue adoptado para escribir toda la librería de .net



Historia.

En el año 2000 Aparece la primera versión en .net el lenguaje adopta todas las bondades de java pero sin necesidad de ejecutar una jvm ya que se ejecuta desde .net, la realidad es que en los últimos años éste lenguaje ha avanzado más que java, adoptando estandares y paradigmas que en java han llegado un poco más tarde.



Instaland.



Instalando.

En Windows podemos utilizar directamente el editor de [Visual Studio Community](#)



Instalando.

En Ubuntu

```
$ wget -q https://packages.microsoft.com/config/ubuntu/18.04/packages-microsoft-prod.deb  
$ sudo dpkg -i packages-microsoft-prod.deb  
  
---  
$ sudo add-apt-repository universe  
---  
  
$ sudo apt-get install apt-transport-https  
$ sudo apt-get update  
$ sudo apt-get install dotnet-sdk-2.2
```



Instalando.

En Mac

Instalamos el paquete predefinido.



Instalando.

Si vamos a utilizar Visual Studio Code, debemos instalar las siguientes extensiones.



C#
ms-vscode  3.4M

C# for Visual Studio Code
(powered by OmniSharp).



.run
Code Runner  1.2M

Run C, C++, Java, JS, PHP, Python, Perl, Ruby, Go...



C# Extensions
jchannon  271.8K

C# IDE Extensions for VSCode



C# XML Documentation..
k--kato  148.6K

Generate C# XML documentation comments...



Que es .Net



Que es .Net

Microsoft .NET es una plataforma de desarrollo y ejecución de aplicaciones. Esto quiere decir que no sólo nos brinda todas las herramientas y servicios que se necesitan para desarrollar modernas aplicaciones empresariales y de misión crítica, sino que también nos provee de mecanismos robustos, seguros y eficientes para asegurar que la ejecución de las mismas sea óptima.



Que es .Net

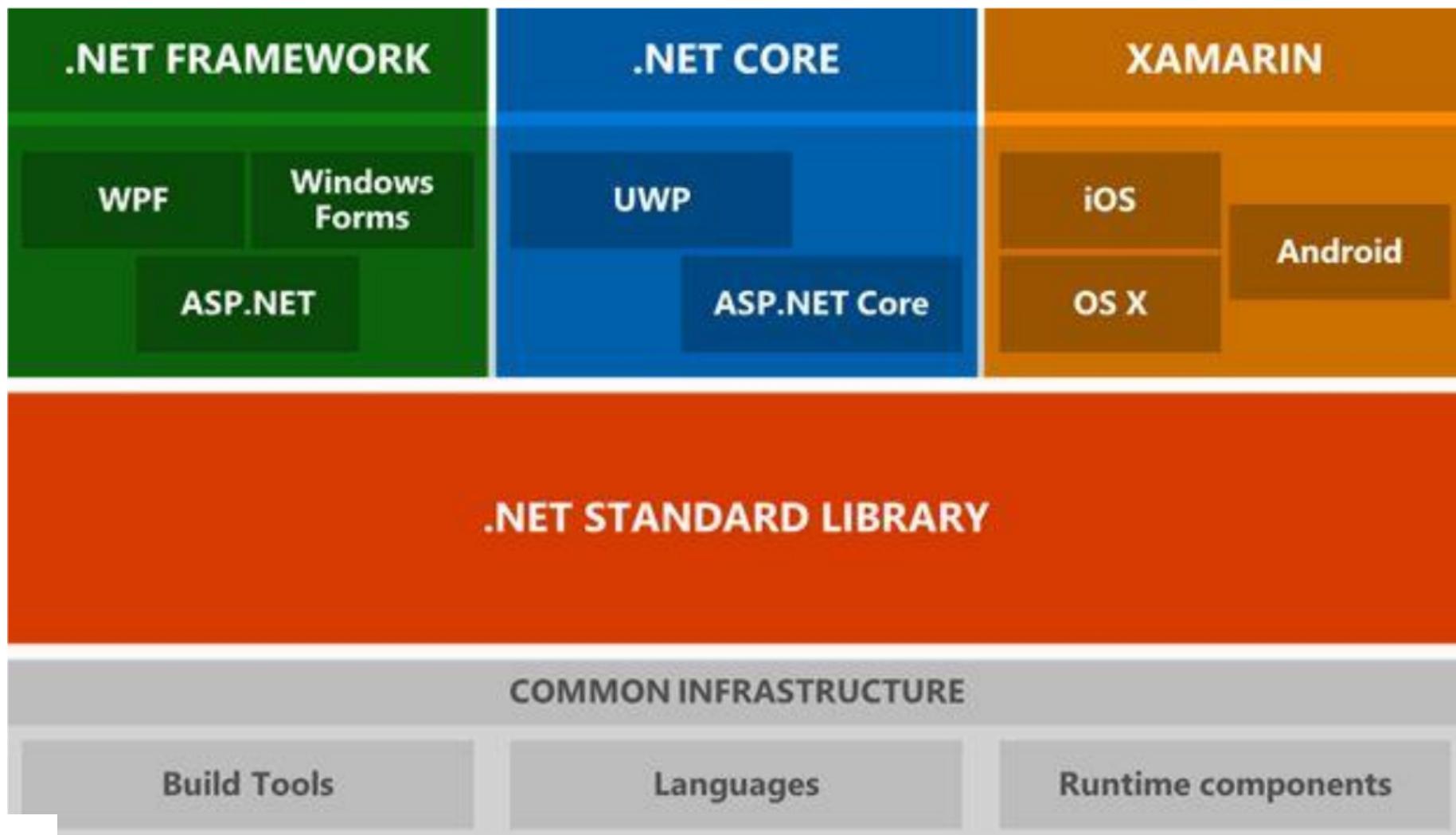
Los componentes principales de la plataforma .NET son:

- Un entorno de ejecución de aplicaciones, también llamado “Runtime”, que es un componente de software cuya función es la de ejecutar las aplicaciones .NET e interactuar con el sistema operativo ofreciendo sus servicios y recursos.
- Un conjunto de bibliotecas de funcionalidades y controles reutilizables, con una enorme cantidad de componentes ya programados listos para ser consumidos por otras aplicaciones.
- Un conjunto de lenguajes de programación de alto nivel, junto con sus compiladores y linkers, que permitirán el desarrollo de aplicaciones sobre la plataforma .NET.
- Un conjunto de utilitarios y herramientas de desarrollo para simplificar las tareas más comunes del proceso de desarrollo de aplicaciones
- Documentación y guías de arquitectura, que describen las mejores prácticas de diseño, organización, desarrollo, prueba e instalación de aplicaciones .NET



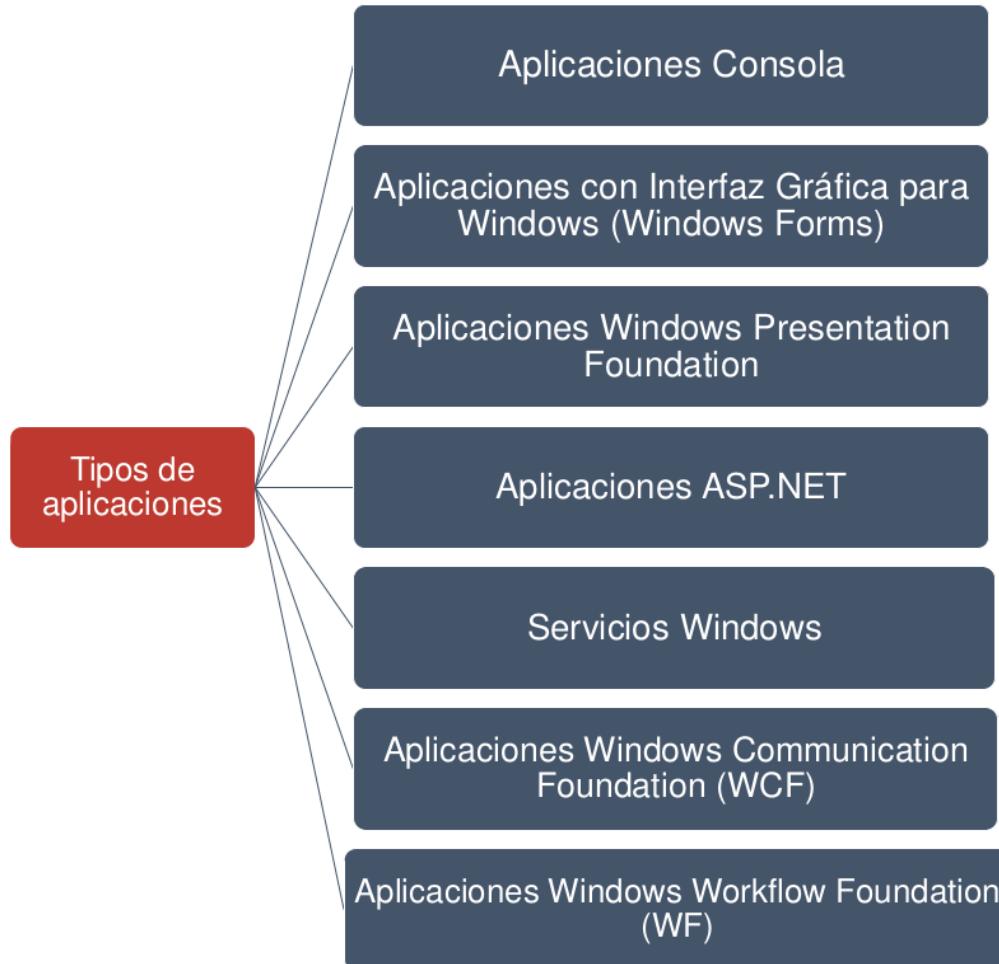
Que es .Net

Este es el ecosistema de .NET



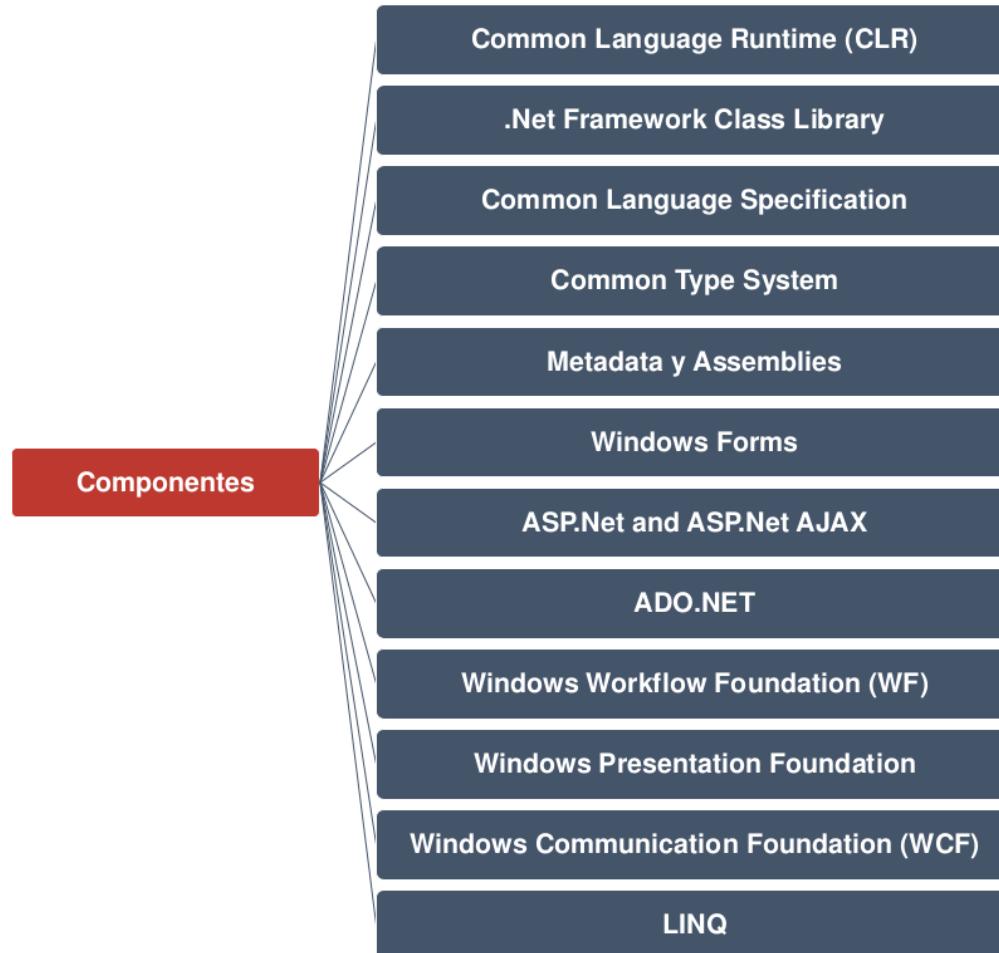
Que es .Net

Tipo de Aplicaciones .NET



Que es .Net

Componentes .NET



Que es .Net Core



Que es .Net Core

A finales de 2015 Microsoft anunció una versión completamente open source de su famoso framework “.NET”. Esta versión, conocida como “.NET core”, destaca no solamente por ser multiplataforma y poder ejecutarse en Windows, Linux, OSX o Android, sino también por ser la versión del framework que mayor atención y desarrollo está recibiendo por parte de Microsoft.



Que es .Net Core

Por ejemplo, la versión de ASP más popular hoy en día es precisamente ASP.NET core MVC. Es decir, ASP sobre .NET core y utilizando el patrón “Model-View-Controller”. Especialmente en el mundo del desarrollo web, Microsoft está primando .NET core por encima del tradicional “.NET Framework” que sólo puede ejecutarse en plataformas Windows.



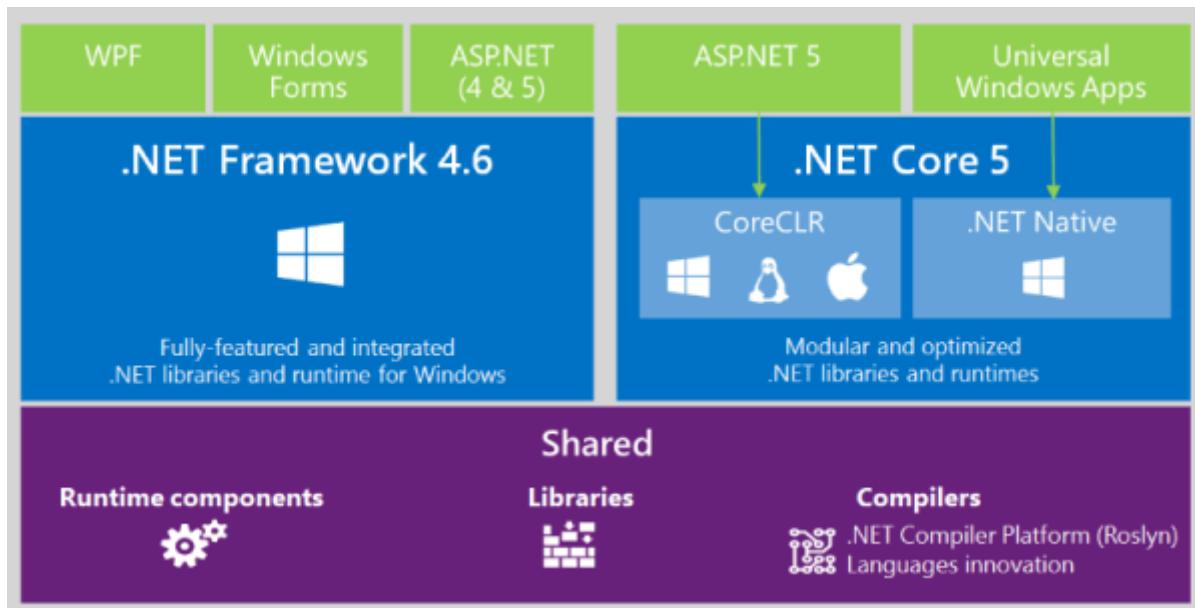
Que es .Net Core

En cuanto a lenguajes de programación, .NET Core soporta de forma nativa tanto C#, F# o Visual Basic .NET, siendo hoy en día C# el lenguaje más popular con diferencia.

En general, .NET Core permite una combinación muy potente de tecnologías que se creían irreconciliables hasta hace poco: la estabilidad y versatilidad de Linux con la potencia de lenguajes como C# y una parte considerable del framework .NET.



Que es .Net Core



Que es CLR



Que es CLR

CLR son las siglas de **Common Language Runtime** o entorno en tiempo de ejecución de lenguaje común. Es un entorno de ejecución para los códigos de los programas que corren sobre la plataforma Microsoft .NET. El CLR es el encargado de compilar una forma de código intermedio llamada Common Intermediate Language (CIL, anteriormente conocido como MSIL, por Microsoft Intermediate Language), al código de máquina nativo, mediante un compilador en tiempo de ejecución. No debe confundirse el CLR con una máquina virtual, ya que una vez que el código está compilado, corre nativamente sin intervención de una capa de abstracción sobre el hardware subyacente

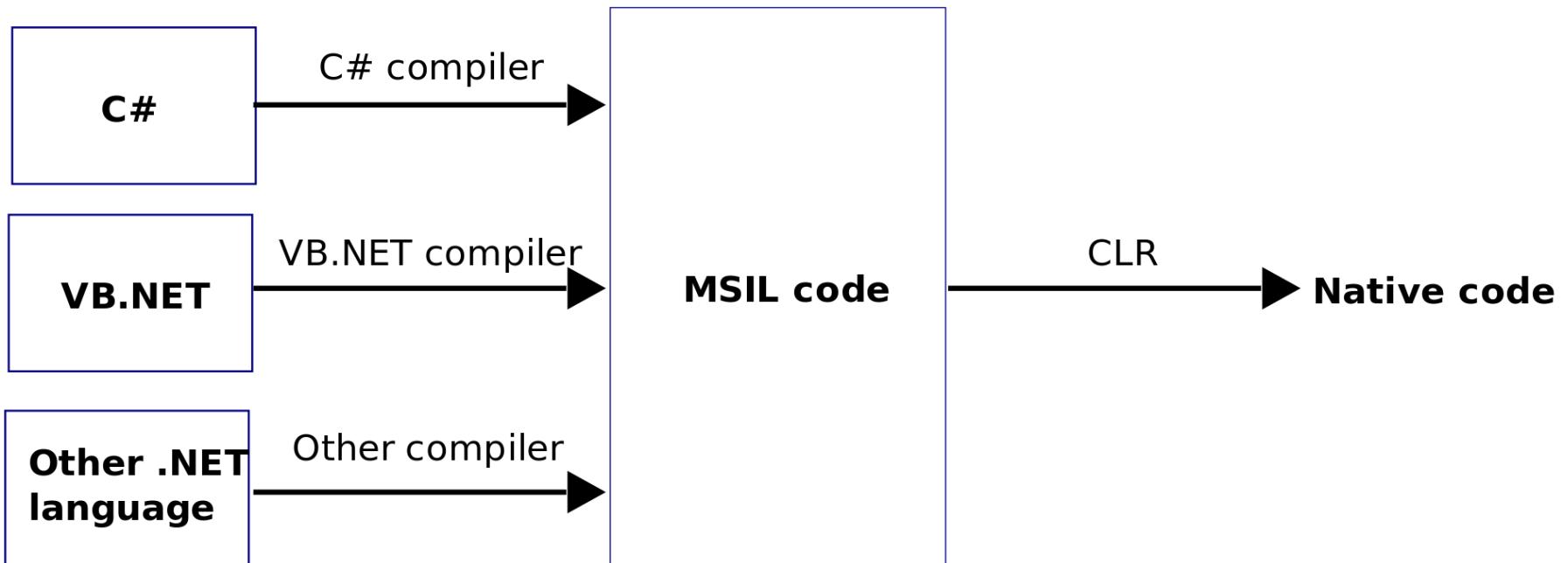


Que es CLR

Source code

Bytecode

Native code



Compile time

Runtime



Creando nuestro primer proyecto.



Creando nuestro primer proyecto.

Comprobamos que tenemos acceso .core para ello ejecutamos en nuestro terminal la siguiente instrucción.

```
$ dotnet --info
```

Nos mostrara la información de .net Core.



Creando nuestro primer proyecto.

Ahora para crear nuestro proyecto simplemente creamos un directorio y escribimos la siguiente instrucción.

```
$dotnet new console
```

Esto nos creará un nuevo proyecto de consola.



Creando nuestro primer proyecto.

Ahora para ejecutarlo sólo debemos ejecutar el siguiente comando

```
$ dotnet run
```

El cual nos sacará por pantalla si todo va bien.

```
$ Hello World!
```

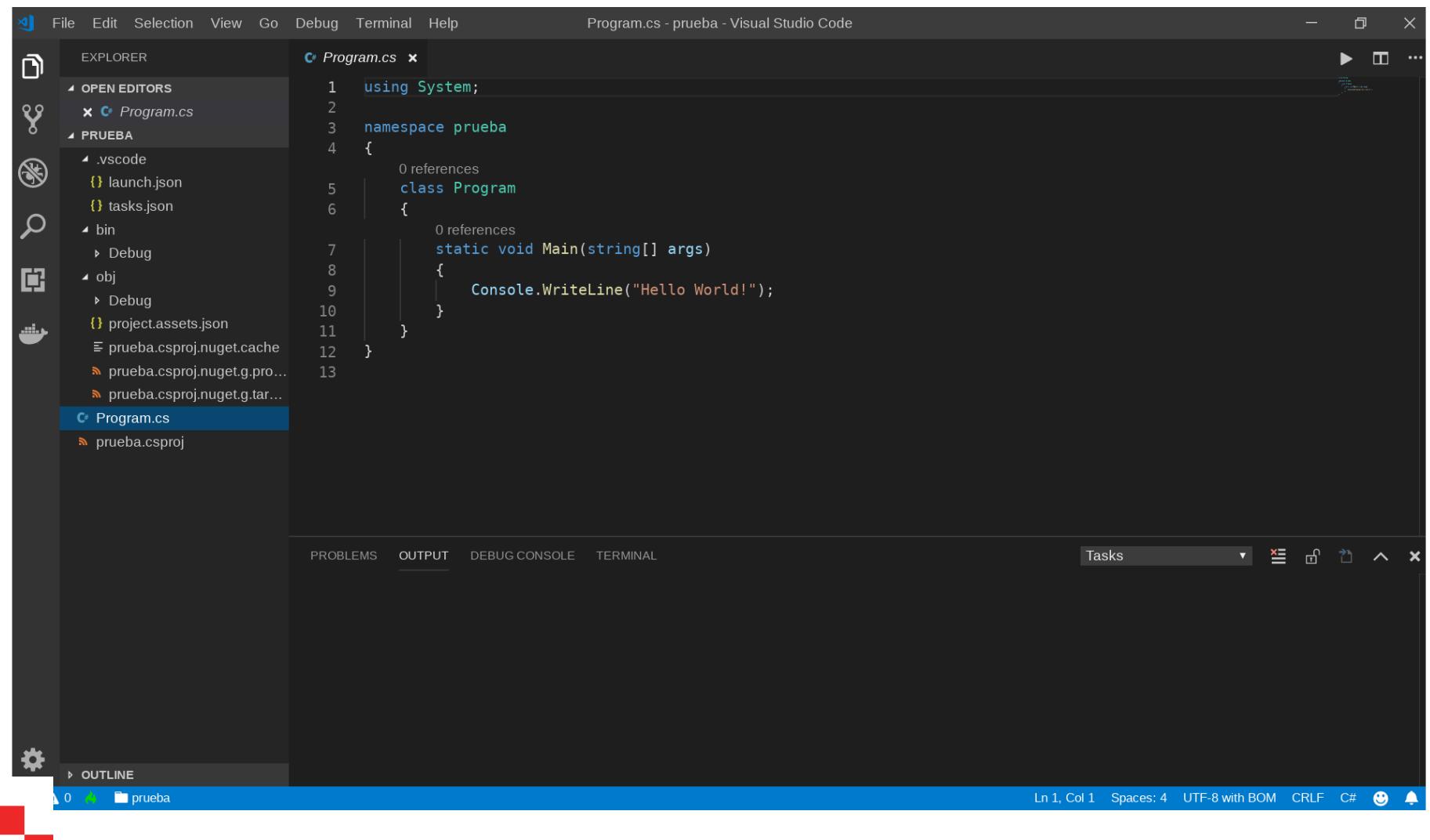


Estructura del proyecto.



Estructura del proyecto.

Ahora vamos a ver la estructura de nuestro proyecto.



The screenshot shows the Visual Studio Code interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Debug, Terminal, Help.
- Title Bar:** Program.cs - prueba - Visual Studio Code.
- Left Sidebar (Explorer):**
 - OPEN EDITORS:** Program.cs (selected).
 - PRUEBA:** A folder containing:
 - .vscode (with launch.json and tasks.json)
 - bin (with Debug)
 - obj (with Debug)
 - project.assets.json
 - prueba.csproj.nuget.cache
 - prueba.csproj.nuget.g.pr...
 - prueba.csproj.nuget.g.tar...
 - Bottom:** Program.cs (selected), prueba.csproj.
- Code Editor:** Displays the following C# code:

```
1  using System;
2
3  namespace prueba
4  {
5      0 references
6      class Program
7      {
8          0 references
9          static void Main(string[] args)
10         {
11             Console.WriteLine("Hello World!");
12         }
13     }
}
```
- Bottom Navigation:** PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL.
- Bottom Status Bar:** Tasks, Ln 1, Col 1, Spaces: 4, UTF-8 with BOM, CRLF, C#, 😊, 📡.

Estructura del proyecto.

Un programa puede crearse empleando varias clases. La forma más sencilla es utilizando una única clase. Para indicarle a c# que clase contiene el programa, rutina o método principal main() y en este se incluyen las sentencias del programa principal. Las sentencias se separan o punto y coma.

```
using System;

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



Estructura del proyecto.

Como cualquier lenguaje de programación, nos permite también añadir comentarios. Estos comentarios se emplean para facilitar la legibilidad del programa, aunque por buenas prácticas no debería ser lo habitual ya que el código debería ser lo suficiente legible, como para no tener que complementar el mismo utilizando comentarios.

Para la definición de un comentario de una sola linea

```
// Este es un comentario de una linea.
```

También podemos comentar diferentes líneas utilizando bloques.

```
/*
 * Comentario de
 * diferentes líneas.
 */
```



Características cSharp



Características cSharp

Ahora vamos a ver algunas características de c#



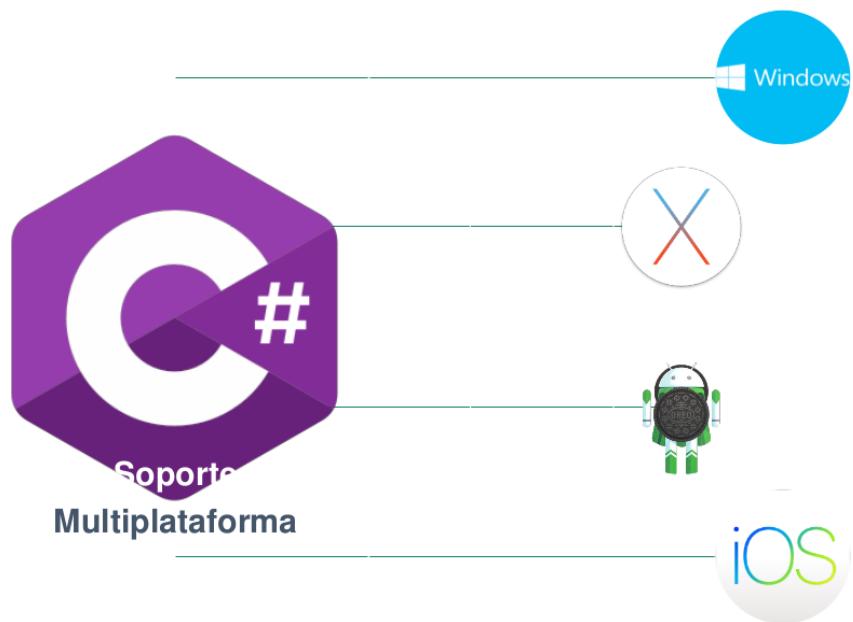
Características cSharp



Características cSharp



Características cSharp



Sintaxis cSharp



Sintaxis cSharp

A continuación vamos a empezar a entender el lenguaje.



I. Variables.



Variables.

La Variables nos sirve para almacenar información de nuestro programa según cambie su valor o no durante la ejecución del programa lo podemos distinguir en:

- **Variables:** Sirven para almacenar los datos y en este caso pueden cambiar n veces durante la ejecución de un programa.
- **Constantes** también sirven para almacenar datos pero una vez asignado el valor, éste no puede modificarse posteriormente.



Variables.



Uso de la Variable

Almacenamiento
en Memoria del
Runtime (Tiempo
de Ejecución)

Duración igual a
la Ejecución de la
Aplicación, Clase
o Método



Asignación.

Para asignar las variables debido a que C# es un lenguaje altamente tipado

[Modificador Acceso] [tipo] [nombre de la variable] = [valor/nulo];

Ejemplo de variables.

```
\Definicion entero.  
int contador;  
\Definición string inicializado.  
String cadena="Hola Mundo";  
\ Definición boolean inicializado.  
boolean flag = true;
```



Asignación

Para definir las constantes en c# se define mediante la palabra reservada **const**

```
[const] [tipo] [nombrConstante] = [valor];
```

Ejemplo de constantes.

```
class Program
{
    const double pi =3.14;
    static void Main(string[] args)
    {
        Console.WriteLine("Valor de pi ->" + pi);
    }
}
```

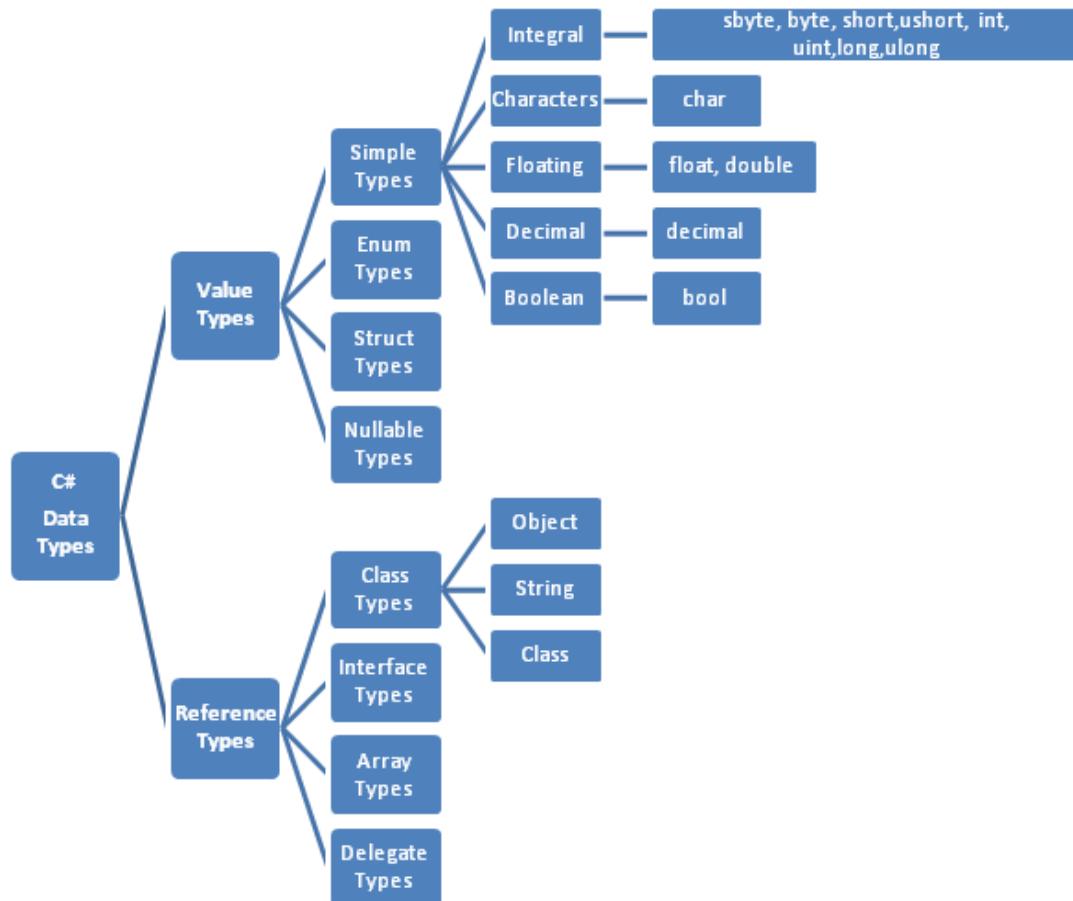


Tipos de datos en cSharp

Hay dos clases de tipos en C#: tipos de valor y tipos de referencia. Las variables de tipos de valor contienen directamente los datos, mientras que las variables de los tipos de referencia almacenan referencias a los datos, lo que se conoce como objetos. Con los tipos de referencia, es posible que dos variables hagan referencia al mismo objeto y que, por tanto, las operaciones en una variable afecten al objeto al que hace referencia la otra variable. Con los tipos de valor, cada variable tiene su propia copia de los datos y no es posible que las operaciones en una variable afecten a la otra (excepto en el caso de las variables de parámetro ref y out).



Tipos de datos en cSharp



Tipos de datos en cSharp

Los datos numéricos.

Tipo	Representa	Rango	Default Value
bool	Valor booleano	True o False	False
byte	Entero sin signo de 8-bit	0 to 255	0
char	Carácter Unicode 16-bit	U +0000 to U +ffff	'\0'
decimal	Valores decimales de precisión de 128 bits con 28-29 dígitos significativos	(-7.9 x 10 ²⁸ to 7.9 x 10 ²⁸) / 10 ^{0 to 28}	0.0M
double	Tipo de coma flotante de doble precisión de 64 bits	(+/-)5.0 x 10 ⁻³²⁴ to (+/-)1.7 x 10 ³⁰⁸	0.0D
float	Tipo de coma flotante de precisión simple de 32 bits	-3.4 x 10 ³⁸ to + 3.4 x 10 ³⁸	0.0F
int	Tipo entero con signo de 32 bits	-2,147,483,648 to 2,147,483,647	0
long	Tipo entero con signo de 64 bits	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	0L
sbyte	Tipo entero con signo de 8 bits	-128 to 127	0
short	Tipo entero con signo de 16 bits	-32,768 to 32,767	0
uint	Tipo entero sin signo de 32 bits	0 to 4,294,967,295	0
ulong	Tipo entero sin signo de 64 bits	0 to 18,446,744,073,709,551,615	0
ushort	Tipo entero sin signo de 16 bits	0 to 65,535	0



Tipos de datos en cSharp

Literales o constants literales. Un literal es una constante cuyo nombre o identificador es la representación escrita de su valor y tienen ya ese significado en el código fuente de un programa en c#. Por ejemplo:

- Una constante literal boolean son false y true
- Otra constante literal de tipo carácter aparece entre comillas simple, un dato de tipo char representa un único carácter.



Tipos de datos en cSharp

Secuencia de Escape Debido a los literales algunos caracteres pueden causar algún problema

Secuencia de escape	Valor
\b	Retroceso o <i>backspace</i> (equivalente a \u0008)
\t	Tabulador (equivalente a \u0009)
\n	Nueva línea (equivalente a \u000A)
\f	Salto de página (equivalente a \u000C)
\r	Retorno de carro (equivalente a \u000D)
\"	Doble comilla (equivalente a \u0022)
'	Comilla simple (equivalente a \u0027)



Tipos de datos en cSharp

Secuencia de Escape Ejemplo:

```
class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("El siguiente texto esta entre \"comillas dobles\" y " +
            "este entre \'comillas simples\'\n Saltamos de linea y
\ttabulamos");
    }
}
```



Conversiones entre tipos.

Otra de las posibilidades interesantes y necesarias es la conversión de uno tipo de dato a otro. Existen cuatro categorías de conversion:

- Conversiones implícitas.
- Conversiones explícitas.
- Conversiones definidas por el usuario.
- Conversiones con clases del asistente.



Conversiones entre tipos.

Conversiones implícitas Las conversiones implícitas son aquellas a las que asigna directamente entre tipos sin que el valor se pueda truncar ni redondearse.

Tipo de origen	Tipo de destino
Byte	short, ushort, int, uint, long, ulong, float, double o decimal
Sbyte	short, int, long, float, double o decimal
Int	long, float, double o decimal
Uint	long, ulong, float, double o decimal
Short	int, long, float, double o decimal
Ushort	int, uint, long, ulong, float, double o decimal
Long	float, double o decimal
Ulong	float, double o decimal
Float	double
Char	ushort, int, uint, long, ulong, float, double o decimal



Conversiones entre tipos.

Conversiones implícitas

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

        int num = 2147483647;
        long bigNum = num;
        Console.WriteLine("Diferencia int ->" + num + " Diferencia bigNum ->" +
bigNum);

    }
}
```



Conversiones entre tipos.

Conversiones explícitas Requieren un operador de conversión. La conversión es necesaria si es posible que se pierda información en la conversión, o cuando es posible que la conversión no sea correcta por otros motivos. Entre los ejemplos típicos están la conversión numérica a un tipo que tiene menos precisión o un intervalo más pequeño, y la conversión de una instancia de clase base a una clase derivada.

Tipo de origen	Tipo de destino
Byte	sbyte o char
Sbyte	byte, ushort, uint, ulong o char
Int	sbyte, byte, short, ushort, uint, ulong o char
Uint	sbyte, byte, short, ushort, int o char
Short	sbyte, byte, ushort, uint, ulong o char
Ushort	sbyte, byte, short o char
Long	sbyte, byte, short, ushort, int, uint, ulong o char
Ulong	sbyte, byte, short, ushort, int, uint, ulong o char
Float	sbyte, byte, short, ushort, int, uint, long, ulong, char o decimal
Double	sbyte, byte, short, ushort, int, uint, long, ulong, char o decimal
Char	sbyte, byte o short
Decimal	sbyte, byte, short, ushort, int, uint, long, ulong, char, float o double



Conversiones entre tipos.

Conversiones explícitas

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

        double x = 1234.7;
        int a;
        // Cast double to int.
        a = (int)x;
        System.Console.WriteLine(a);

    }
}
```



Conversiones entre tipos.

Conversiones definidas por el usuario las conversiones definidas por el usuario se realizan por medio de métodos especiales que se pueden definir para habilitar las conversiones explícitas e implícitas entre tipos personalizados que no tienen una relación de clase base-clase derivada.



Conversiones entre tipos.

Conversiones con clases del asistente para realizar conversiones entre tipos no compatibles, como enteros y objetos System.DateTime, o cadenas hexadecimales y matrices de bytes puede usar la clase System.BitConverter, la clase System.Convert y los métodos Parse de los tipos numéricos integrados, como Int32.Parse



Strings

Un string no es más que una cadena de texto compuesto por una sucesión de un conjunto de caracteres alfanuméricos signos de puntuación y espacios en blanco con más o menos sentidos. Para crear una cadena de tipo String hay que instanciar la misma.

```
class Program
{
    static void Main(string[] args)
    {
        string nombre = "Cadena de Texto";
        Console.WriteLine(nombre);

    }
}
```



Strings

Al tratarse de un objeto de tipo String podemos realizar una serie de operaciones, por ejemplo el operado de concatenación +

```
class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Hola" + "Mundo");
        Console.WriteLine("Año"+1215);
        Console.WriteLine("x"+3+2);
        Console.WriteLine(3+2+"x");
    }
}
```



Strings

Método Substring, obtiene una subcadena de texto a partir de otra cadena más grande.

```
class Program
{
    static void Main(string[] args)
    {
        string nombre = "Cadena de Texto";
        Console.WriteLine(nombre.Substring (7, 2));
    }
}
```



Strings

Método **IndexOf**, obtiene una posición dentro de una cadena de texto a partir de otro string.

```
class Program
{
    static void Main(string[] args)
    {
        string nombre = "Cadena de Texto";
        Console.WriteLine(nombre.IndexOf ("de"));
        //También a partir de un índice dado.
        Console.WriteLine(nombre.IndexOf ("de", 3));
    }
}
```



Strings

Métodos ToUpper() y ToLower(), convierte a mayúsculas y minúsculas una cadena de texto dado.

```
class Program
{
    static void Main(string[] args)
    {
        string nombre = "Javier Rodriguez";
        Console.WriteLine(nombre.ToUpper());
        //También a partir de un índice dado.
        Console.WriteLine(nombre.ToLower());
    }
}
```



Strings

Métodos `.StartsWith()` y `.EndsWith()`, comprueba si una cadena de caracteres empieza y acaba por una subcadena.

```
class Program
{
    static void Main(string[] args)
    {
        string nombre = "Javier Rodriguez";
        Console.WriteLine(nombre.StartsWith("Javier"));
        //También a partir de un índice dado.
        Console.WriteLine(nombre.EndsWith("Rodriguez")));
    }
}
```



Strings

Métodos `.TrimStart()`, `.TrimEnd()` y `.Trim()` elimina los caracteres del inicio del final o lo que les indiquemos.

```
class Program
{
    static void Main(string[] args)
    {
        string sentence = "The dog had a bone, a ball, and other toys.";
        char[] charsToTrim = {',', '.', ' '};
        string[] words = sentence.Split();
        foreach (string word in words)
            Console.WriteLine(word.TrimEnd(charsToTrim));
    }
}
```



Strings

Métodos .PadLeft() y .PadRight() añade elementos a la cadena del carácter y a la izquierda.

```
class Program
{
    static void Main(string[] args)
    {
        string nombre = "Gonzalez";
        nombre = nombre.PadRight(10, 'XX');
        Console.WriteLine(nombre);
        string codigo = "123";
        codigo = codigo.PadLeft(6, '0');
        Console.WriteLine(codigo);
    }
}
```



Strings

Métodos `.Join()` y `.Split()` nos añade o divide a partir de un carácter.

```
class Program
{
    static void Main(string[] args)
    {
        string str = "This is our Demo String";
        var arr = str.Split(' ');
        string rest = string.Join(" ", arr.Skip(1));
        Console.WriteLine(rest);
    }
}
```



Strings

Método `.format()` nos permite formatear y bindear la información que pasamos.

```
class Program
{
    static void Main(string[] args)
    {
        string prueba = String.Format("Formato de la cadena con un {0} parámetro. Ahora son las {1}. Valor numérico: {2}", 1, DateTime.Now, 15.5);
        Console.WriteLine(prueba);
        Console.ReadKey();
    }
}
```



Strings

Método `.replace()` nos permite remplazar una subcdena por otra.

```
class Program
{
    static void Main(string[] args)
    {
        string texto = "Aprendiendo CSharp";
        string textoModificado = texto.Replace(" CSharp ", " C#");
        Console.WriteLine(textoModificado);
        Console.ReadKey();
    }
}
```



Strings

Método .Length nos devuelve el número de caracteres que contiene el String.

```
class Program
{
    static void Main(string[] args)
    {
        string texto = "Aprendiendo CSharp";
        Console.WriteLine(texto.Length);
        Console.ReadKey();
    }
}
```



Los Arrays

Los arrays o vectores son objetos también, y permite representar una secuencia lineal y finita de elementos del mismo tipo. Estos elementos pueden ser de tipo primitivos o perteneciente a otras clases. Al tratarse de un objeto tiene también características y métodos propios. Para declarar una referencia a un array se realiza de la siguiente forma:

```
tipo[] nombreArray = new tipo[tamano_array];  
Int [] array = new int [10];
```



Los Arrays

Podemos insertar un elemento en el array a partir de un índice del mismo, así como obtener el valor del mismo.

```
class Program
{
    static void Main(string[] args)
    {
        int[] array = new int[50];
        for (int i = 0; i < 50; i++)
        {
            array[i] = i + 1;
            Console.WriteLine("Indice ->" + i + " Valor ->" + array[i] );
        }
    }
}
```



Los Tipos Enumerados

Un tipo enumerado nos permite restrigir los posibles valores que puede tomar una variable. Esto ayuda a reducir los errores en el código y permite algunos usos especiales interesantes. Se define de la siguiente forma:

```
enum Day { Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday };  
enum Month : byte { Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec };  
class Program  
{  
    static void Main(string[] args)  
    {  
        Day today = Day.Monday;  
        int dayNumber =(int)today;  
        Console.WriteLine("{0} is day number #{1}.", today, dayNumber);  
        Month thisMonth = Month.Dec;  
        byte monthNumber = (byte)thisMonth;  
        Console.WriteLine("{0} is month number #{1}.", thisMonth,  
monthNumber);  
    }  
}
```



Los Tipos Nullables

Los tipos por referencia son nullables por definición. ¿Qué implica que sean nullables? pues nada más y nada menos que puedan tener un valor null o en términos prácticos, ningún valor. Esto se puede explicar desde el nombre mismo, al ser referencias estas pueden estar referenciando a un valor o a la nada (a null).

Por otro lado, los tipos por valor siempre tienen un valor (valga la redundancia) asignado. Es así que un int, bool, double, etcétera nunca podrán valer null. Y es que estos no hacen referencia a ningún valor, sino que son el valor mismo.

Muchas veces esta limitante nos lleva a inventarnos números mágicos como por ejemplo usar el -1 para decir que un entero no tiene valor, o lógicas extrañas dentro de nuestro programa como decir que si un DateTime vale DateTime.MinValue es porque no tiene valor. También nos complica un poco la vida cuando se trabaja con bases de datos en donde los datos pueden tener un valor o estar indefinidos.



Los Tipos Nullables

Operadores ? y ?? con los tipos Nullables. Al operador ?? se le llama el operador de uso combinado de NULL. Dicho operador devuelve el operando izquierdo si no es NULL; de lo contrario, devuelve el operando derecho.

```
/ x puede ser un int32 o nulo
int? x = null;
// y puede ser True / False o Nulo
Bool? y = null;
// z puede ser un Double o Nulo
doble? z = null;
```



Los Tipos Nullables

El método `.HasValue()` devuelve el valor válido, por ejemplo

```
// x puede ser un int32 o nulo
int? x = null;
(x.HasValue)
// x contiene un valor entero válido
```



Los Tipos Nullables

El método `.Value` obtiene el valor del tipo.

```
class Program
{
    static void Main(string[] args)
    {
        int? x = null;
        // y es un valor Int32
        int y = 0;
        try
        {
            y = x.Value;
        } catch (InvalidOperationException)
        {
            Console.WriteLine ("Operación no válida!");
        }
    }
}
```



Los Tipos Nullables

El método `GetValueOrDefault()` devuelve el valor o el existente por defecto sino dispone.

```
class Program
{
    static void Main(string[] args)
    {
        // x puede ser un int32 o nulo
        int? x = null;
        // y recibirá valor 0 (cero) que es el valor predeterminado para el
        tipo Int32
        int y = x.GetValueOrDefault ();
        Console.WriteLine('El valor de y'+ y);
    }
}
```



II. Operadores



Operadores

Un operador lleva a cabo operaciones sobre uno (operador unario), dos (operador binario) o tres (operador ternario) datos u operandos de tipo primitivo devolviendo un valor determinado también de un tipo primitivo. El tipo de valor devuelto tras la evaluación depende del operador y del tipo de los operandos.



Operadores Aritméticos.

C# dispone de diversos operadores aritméticos para los datos numéricos enteros y reales.

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
-	operador unario de cambio de signo	-4	-4
+	Suma	2.5 + 7.1	9.6
-	Resta	235.6 - 103.5	132.1
*	Producto	1.2 * 1.1	1.32
/	División (tanto entera como real)	0.050 / 0.2 7 / 2	0.25 3
%	Resto de la división entera	20 % 7	6



Operadores Aritméticos.

Los operadores aritméticos incrementales son operadores unarios (un único operando). El operando puede ser numérico o de tipo char y el resultado es del mismo tipo que el operando. Estos operadores pueden emplearse de dos formas dependiendo de su posición con respecto al operando.

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
++	Incremento <code>i++</code> primero se utiliza la variable y luego se incrementa su valor <code>++i</code> primero se incrementa el valor de la variable y luego se utiliza	<code>4++</code> <code>a=5;</code> <code>b=a++;</code> <code>a=5;</code> <code>b=++a;</code>	<code>5</code> <code>a vale 6 y b vale 5</code> <code>a vale 6 y b vale 6</code>
--	decremento	<code>4--</code>	<code>3</code>



Operadores Aritméticos Combinados.

Combinan un operador aritmético con el operador de asignación.

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
<code>+=</code>	Suma combinada	<code>a+=b</code>	<code>a=a+b</code>
<code>-=</code>	Resta combinada	<code>a-=b</code>	<code>a=a-b</code>
<code>*=</code>	Producto combinado	<code>a*=b</code>	<code>a=a*b</code>
<code>/=</code>	División combinada	<code>a/=b</code>	<code>a=a/b</code>
<code>%=</code>	Resto combinado	<code>a%=b</code>	<code>a=a%b</code>



Operadores Lógicos Booleanos.

Realizan operaciones de tipo booleano y devuelve un valor booleanos

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
!	Negación - NOT (unario)	<code>!false !(5==5)</code>	true false
	Suma lógica – OR (binario)	<code>true false (5==5) (5<4)</code>	true true
^	Suma lógica exclusiva – XOR (binario)	<code>true ^ false (5==5) ^ (5<4)</code>	true true
&	Producto lógico – AND (binario)	<code>true & false (5==5) & (5<4)</code>	false false
	Suma lógica con cortocircuito: si el primer operando es true entonces el segundo se salta y el resultado es true	<code>true false (5==5) (5<4)</code>	true true
&&	Producto lógico con cortocircuito: si el primer operando es false entonces el segundo se salta y el resultado es false	<code>false && true (5==5) && (5<4)</code>	false false

Operadores de Relación.

Realizan comparaciones entre datos compatibles y devuelven un resultado booleano

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
<code>==</code>	igual que	<code>7 == 38</code>	<code>false</code>
<code>!=</code>	distinto que	<code>'a' != 'k'</code>	<code>true</code>
<code><</code>	menor que	<code>'G' < 'B'</code>	<code>false</code>
<code>></code>	mayor que	<code>'b' > 'a'</code>	<code>true</code>
<code><=</code>	menor o igual que	<code>7.5 <= 7.38</code>	<code>false</code>
<code>>=</code>	mayor o igual que	<code>38 >= 7</code>	<code>true</code>



Operador Condicional.

Es el operador ternario y se expresa de la siguiente forma.

```
valor = ([expresion-logica] ? [Si el resultado es true] [Expresión a ejecutar]
: [Si el resultado es false] [Expresión a ejecutar]);
```

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
?:	operador condicional	a = 4; b = a == 4 ? a+5 : 6-a; b = a > 4 ? a*7 : a+8;	b vale 9 b vale 12

Operadores de Bit.

Tienen operando de tipo entero o (char) y un resultado de tipo entero.

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
<code>~</code>	Negación ó complemento binario (unario)	<code>~12</code>	-13
<code> </code>	Suma lógica binaria – OR (binario)	<code>12 10</code>	8
<code>^</code>	Suma lógica exclusiva – XOR (binario)	<code>12 ^ 10</code>	6
<code>&</code>	Producto lógico binario – AND (binario)	<code>12 & 10</code>	14
<code><<</code>	Desplaza a la izquierda los bits del 1º operando tantas veces como indica el 2º operando (por la derecha siempre entra un cero)	<code>7<<2</code> <code>-7<<2</code>	28 -28
<code>>></code>	Desplaza a la derecha los bits del 1º operando tantas veces como indica el 2º operando (por la izquierda entra siempre el mismo bit más significativo anterior)	<code>7>>2</code> <code>-7>>2</code>	1 -2
<code>>>></code>	Desplaza a la derecha los bits del 1º operando tantas veces como indica el 2º operando – sin signo (por la izquierda entra siempre un cero).	<code>7>>>2</code> <code>-7>>>2</code>	1 1073741822



Prioridad entre Operadores.

C# establece un orden de prioridad respecto a la forma de ejecutar los operadores, estos si tienen la misma prioridad se evalúan de izquierda a derecha dentro de la expresión.



Prior.	Operador	Tipo de operador	Operación
1	<code>++</code> <code>--</code> <code>+, -</code> <code>~</code> <code>i</code>	Aritmético Aritmético Aritmético Integral Booleano	Incremento previo o posterior (unario) Incremento previo o posterior (unario) Suma unaria, Resta unaria Cambio de bits (unario) Negación (unario)
2	<code>(tipo)</code>	Cualquiera	
3	<code>*, /, %</code>	Aritmético	Multiplicación, división, resto
4	<code>+, -</code> <code>+</code>	Aritmético Cadena	Suma, resta Concatenación de cadenas
5	<code><<</code> <code>>></code> <code>>>></code>	Integral Integral Integral	Desplazamiento de bits a izquierda Desplazamiento de bits a derecha con inclusión de signo Desplazamiento de bits a derecha con inclusión de cero
6	<code><, <=</code> <code>>, >=</code> <code>instanceof</code>	Aritmético Aritmético Objeto, tipo	Menor que, Menor o igual que Mayor que, Mayor o igual que Comparación de tipos
7	<code>==</code> <code>i=</code> <code>==</code> <code>i=</code>	Primitivo Primitivo Objeto Objeto	Igual (valores idénticos) Desigual (valores diferentes) Igual (referencia al mismo objeto) Desigual (referencia a distintos objetos)
8	<code>&</code> <code>&</code>	Integral Booleano	Cambio de bits AND Producto booleano
9	<code>^</code> <code>^</code>	Integral Booleano	Cambio de bits XOR Suma exclusiva booleana
10	<code> </code> <code> </code>	Integral Booleano	Cambio de bits OR Suma booleana
11	<code>&&</code>	Booleano	AND condicional
12	<code> </code>	Booleano	OR condicional
13	<code>? :</code>	Booleano, cualquiera, cualquiera	Operador condicional (ternario)
14	<code>=</code> <code>*=, /=, %=</code> <code>+=, -=</code> <code><<=, >>=</code> <code>>>>=</code> <code>&=, ^=, =</code>	Variable, cualquiera	Asignación Asignación con operación

III. Expresiones, ordenes y bloques.



Expresiones.

Las expresiones son combinaciones de operadores y operandos que se evalúa generándose un único resultado de un tipo determinado.

```
int valor =1;  
if (valor1 > valor2) { }
```

Cuando tenemos expresiones de evaluación complejas es recomendable utilizar paréntesis.

```
//La misma expresión.  
2 +10/5;  
(2+10) / 5;  
2 +(10/5);
```



Sentencias.

Un sentencia es la unidad mínima de ejecución de un programa, al final de cada uno de ellas acaban con ;

```
\\"Sentencias de declaración.  
int numero =3;  
\\"Sentencias de asignación.  
numero =4 ;  
\\"Sentencias de incremento o decremento.  
valor++;
```



Sentencias if-else

Es una bifurcación o sentencia condicional de una o dos ramas. La sentencia de control evalúa la condición lógica o booleana. Si esa condición es cierta entonces se ejecuta la sentencia o senecencias que se encuentra a continuación. En caso contrario, se ejecuta la sentencia que sigue a else La expresión lógica de ir entre paréntesis. Las llaves son obligatorias si las sentencias son compuestas.

```
if(3>1)
{
    System.out.println("3 es Mayor que 1");
}
{
    System.out.println("3 es Menor que 1");
}
```



Sentencias Switch

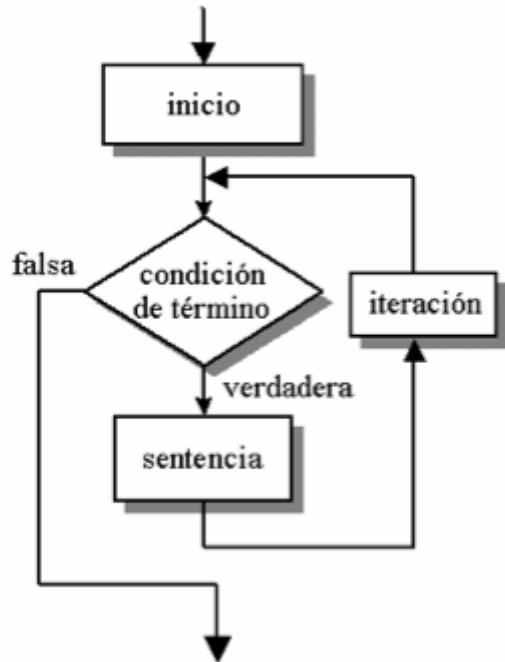
Es una sentencia que no permite modificar la línea de ejecución. En vez de usar un expresión booleana, usa una expresión byte, short, chart, int o un tipo enumerado, también se puede utilizar un String.

```
switch(expresión) {  
    case <valor>:  
        //Sentencia o sentencias  
        break;  
    case <valor>:  
        //Sentencia o sentencias  
        break;  
    default:  
        //Sentencia o sentencias  
        break;
```



Sentencia For

Las sentencias for son bucles o sentencias repetivas que cumplen las siguientes condiciones:



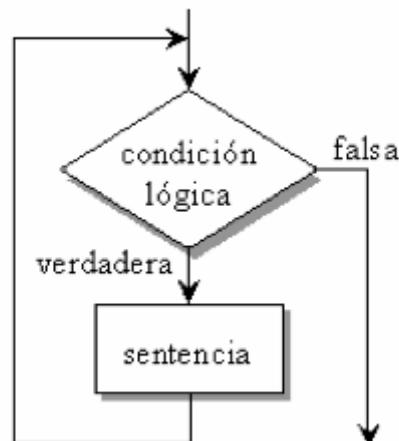
Definición:

```
for( inicio; condicion; iterador)
{
    //Sentencia o sentencias
}
```



Sentencia While

Son sentencias repetitivas con una condición a principio. Se ejecuta una sentencia



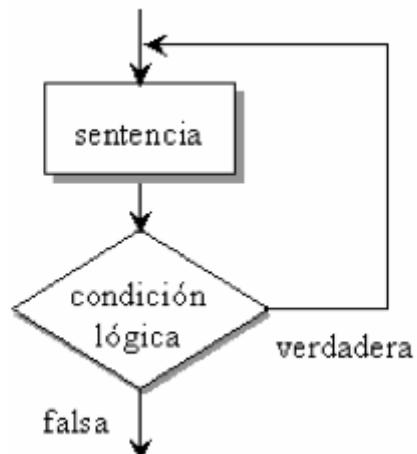
Definición:

```
while(condición)
{
    //Sentencia o sentencias
}
```



Sentencia do-while

Las sentencia do-while son repetitivas con una condición final. se ejecuta mientras sea cierta la condición.



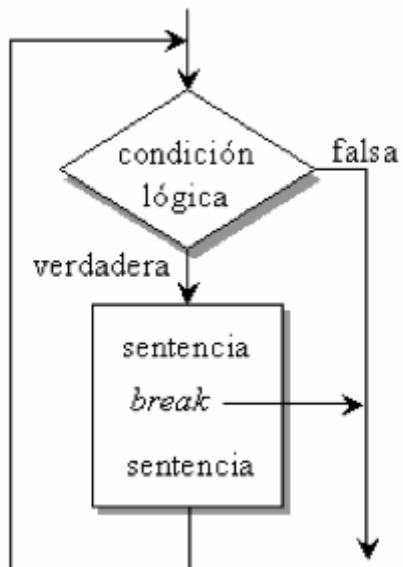
Definición:

```
do
{
    //Sentencia o sentencias
}while(condición)
```



Sentencia Break

La sentencia break se puede utilizar con la sentencia switch o en bucles. Al ejecutarse deja el ámbito de la sentencia en la que se encuentra y pasa a la siguiente sentencias.



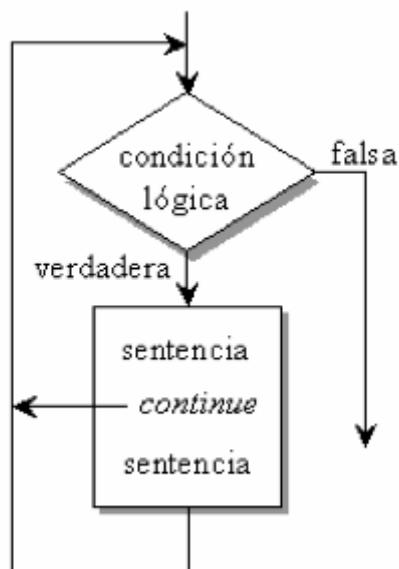
Definición:

```
sentencia;  
break;
```



Sentencia Continue

La sentencia continue se emplea sólo en bucles. Al ejecutarse la iteración en la que se encuentra, el bucle finaliza y se inicia la siguiente.



Definición:

```
sentencia;  
continue;
```



Bloques

Un bloque es un conjunto de sentencias los cuales están delimitados por llaves.

```
if(expresion) {  
    //Bloque1  
} else {  
    //Bloque2  
}
```



Separadores

Sirven para establecer los diferentes bloques y expresiones dispone de diferentes separadores.

Separador	Descripción
()	Permiten modificar la prioridad de una expresión , contener expresiones para el control de flujo y realizar conversiones de tipo . Por otro lado pueden contener la lista de parámetros o argumentos, tanto en la definición de un método como en la llamada al mismo.
{ }	Permiten definir bloques de código y ámbitos y contener los valores iniciales de las variables array
[]	Permiten declarar variables de tipo array (vectores o matrices) y referenciar sus elementos
;	Permite separar sentencias
,	Permite separar identificadores consecutivos en la declaración de variables y en las listas de parámetros. También se emplea para encadenar sentencias dentro de un bucle for
.	Permite separar el nombre de un atributo o método de su instancia de referencia. También separa el identificador de un paquete de los de los subpaquetes y clases

Métodos.

Un método es parte del código que puede ser llamado o invocado por el programa principal o por otro método para realizar alguna tarea específica. El método es llamado por su nombre o identificador seguido por una secuencia de parámetros o argumentos entre paréntesis. Cuando el método finaliza sus operaciones, devuelve habitualmente un valor simple al programa que lo llama.

```
public void setNombre (String nombre)
{
    this.Nombre = nombre;
}
public String getNombre ()
{
    return this.Nombre;
}
```



Métodos.

En C#, los métodos suelen ir asociados con los objetos o instancias en particular lo que operan. Los métodos que no necesitan trabajar con objetos, se denominan métodos estáticos o de clase y se utiliza la palabra reservada static

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

        int num = 2147483647;
        long bigNum = num;
        Console.WriteLine("Diferencia int ->" + num + " Diferencia bigNum ->" +
bigNum);

    }
}
```

