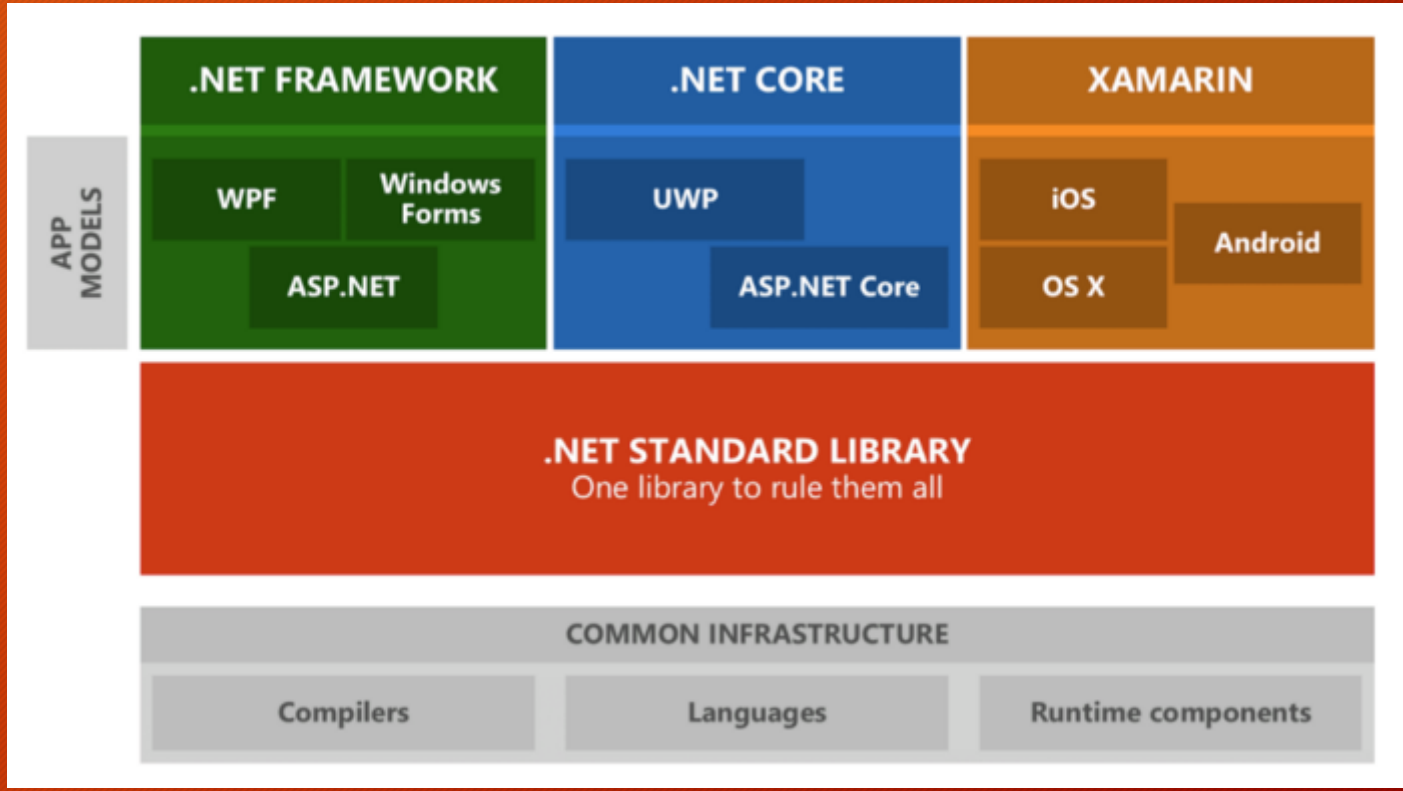
Introducción a .NET

1

Programación II y Laboratorio de Computación II

¿Qué es dotNET?

- Es el stack de tecnologías de Microsoft.
- Consiste de cuatro implementaciones:
  - .NET Framework
  - .NET Core
  - Mono
  - UWP
- Comparten una especificación en común:
  - .NET Standard.
- Comparten una serie de herramientas e infraestructura en común:
  - Lenguajes y sus compiladores: C#, F#, VB.NET.
  - El sistema de proyectos.
  - El NuGet Manager: Administrador de paquetes.



.NET Stack

- Un “stack” es una serie de tecnologías de programación que se utilizan en conjunto para construir y ejecutar aplicaciones.

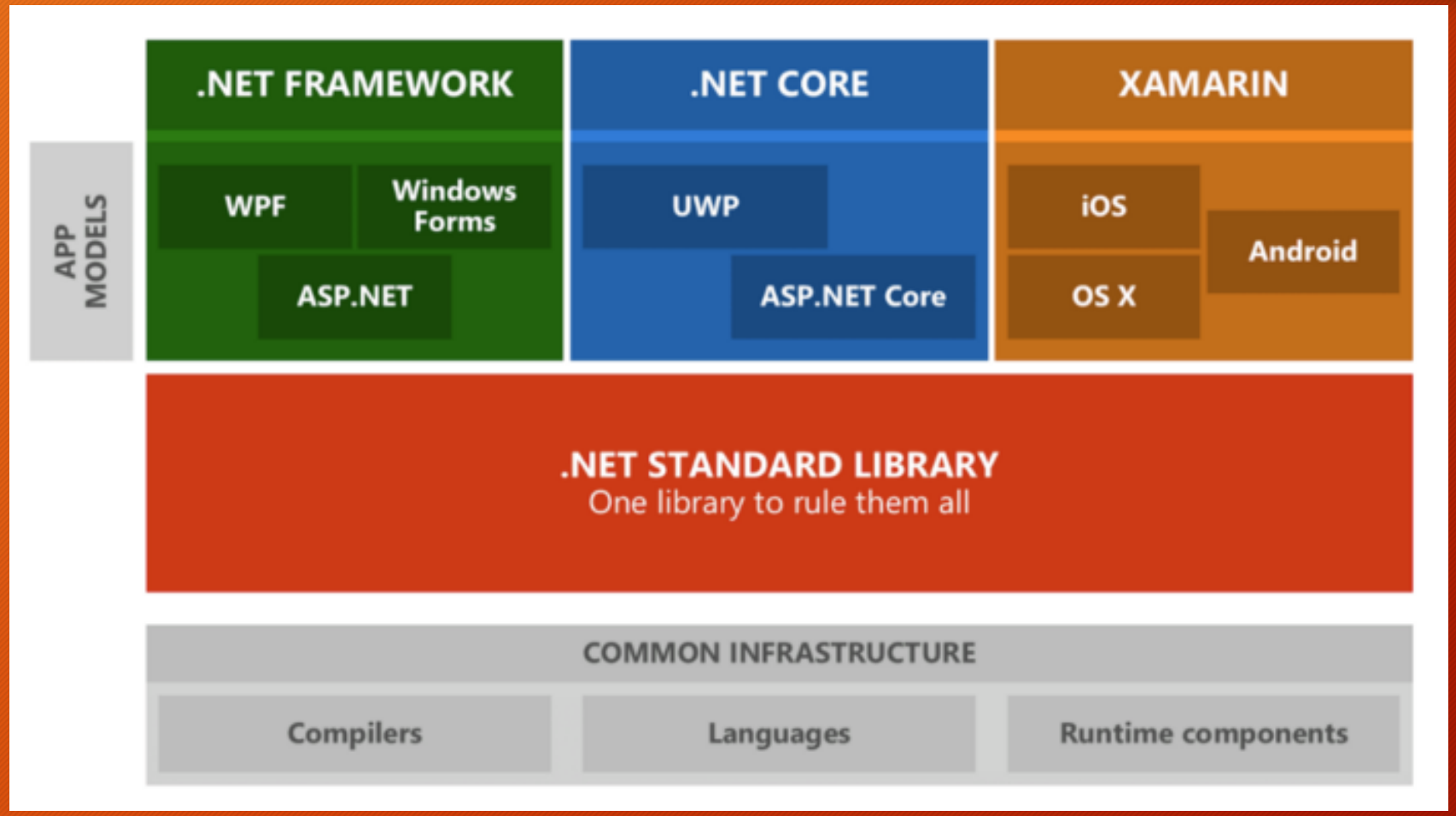
- ".NET stack" es el conjunto formado por .NET Standard y todas las implementaciones de .NET (.NET Framework, .NET Core, etc).

.NET Standard

- Es una serie de contratos y especificaciones que deben cumplir todas las implementaciones de .NET.

- Permite portabilidad entre las distintas implementaciones.
  - El código creado bajo las especificaciones de .NET Standard puede ejecutarse bajo cualquier implementación de .NET que soporte esa versión.

.NET Stack


.NET FRAMEWORK
.NET CORE
XAMARIN
APP MODELS
WPF
Windows Forms
ASP.NET
UWP
ASP.NET Core
iOS
Android
OS X
.NET STANDARD LIBRARY
One library to rule them all
COMMON INFRASTRUCTURE
Compilers
Languages
Runtime components

Implementaciones de .NET

- Las cuatro principales implementaciones de .NET que Microsoft desarrolla y mantiene actualmente son:
  - .NET Core
  - .NET Framework ***<- Este vamos a usar nosotros.***
  - Mono
  - Universal Windows Platform (UWP)

.NET Framework

- Es una plataforma de desarrollo para construir aplicaciones para la web, Windows, Windows Phone, Windows Server y Microsoft Azure.

- Consiste de dos componentes principales:
  - Common Language Runtime (CLR).
  - Biblioteca de clases de .NET Framework.

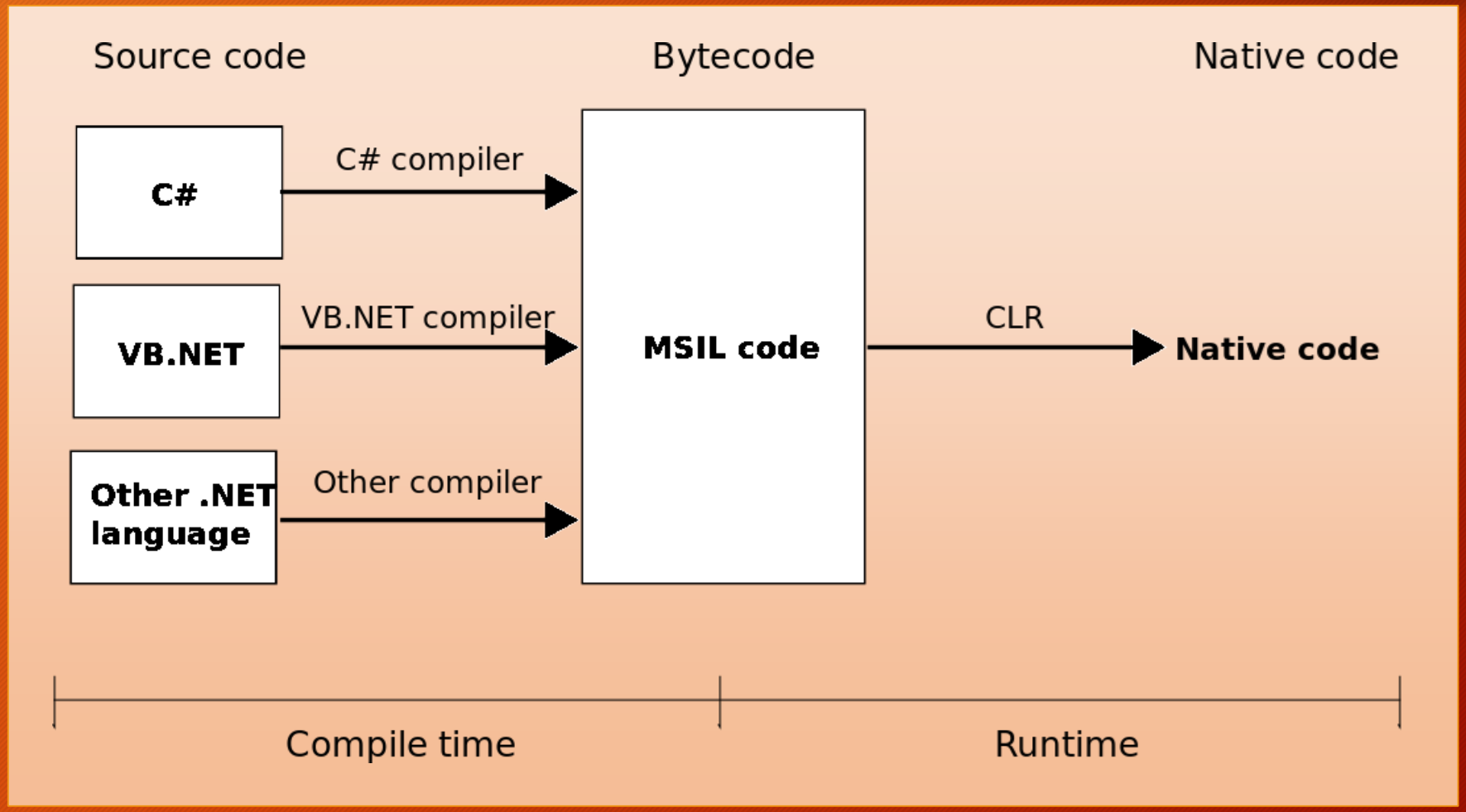
Terminología: Runtime

- Runtime es el **entorno de ejecución** de un programa.
- El sistema operativo es parte del entorno de ejecución pero no es parte del .NET Runtime.
- Ejemplos de entornos de ejecución para .NET:
  - Common Language Runtime (CLR) para .NET Framework.
  - Core Common Language Runtime (CoreCLR) para .NET Core.
  - .NET Native para Universal Windows Platform.
  - Mono runtime para Xamarin.iOS, Xamarin.Android, Xamarin.Mac, y Mono desktop framework.

Common Language Runtime (CLR)

- Es el entorno de ejecución de .NET Framework.
- Maneja la administración y alocación de memoria.
- Sólo para Windows.
- Es también una máquina virtual que ejecuta aplicaciones y compila **lenguaje intermedio** a lenguaje máquina durante la ejecución de la aplicación (just-in-time / JIT).
  - Lenguaje intermedio (IL): Los lenguajes de alto nivel de .NET (como C#) se compilan a un código agnóstico del hardware.
  - Se lo conoce como CIL: Common Intermediate Language. (Similar al BYTECODE de JAVA)

¿Cómo se compila el código en .NET Framework?



Terminología: API

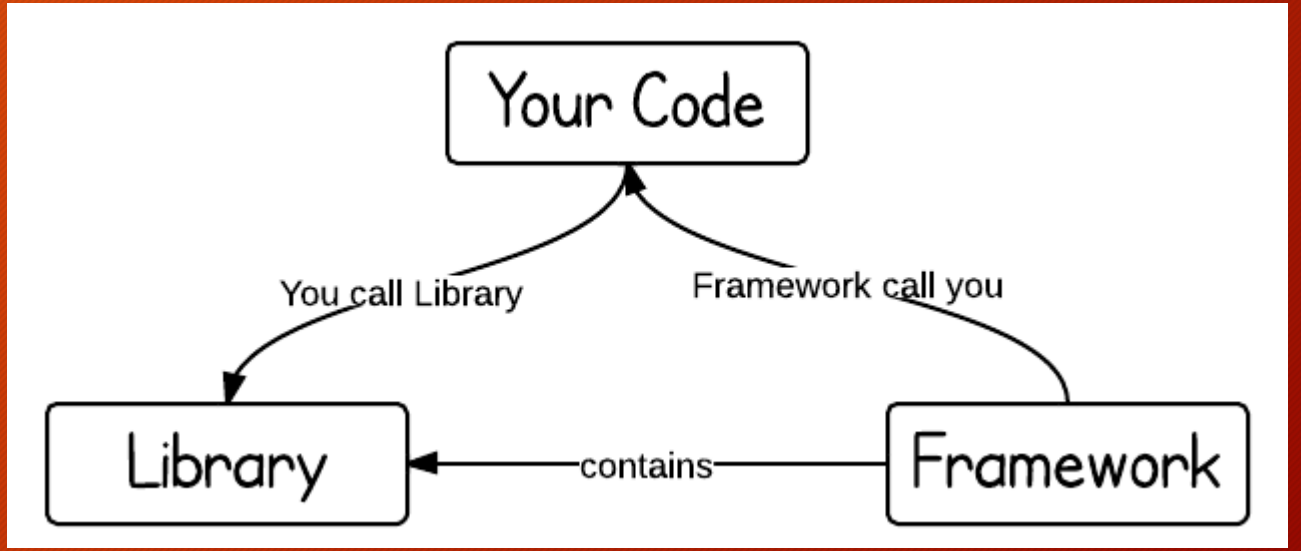
- Acrónimo para Application Programming Interface.
- Expone una serie de operaciones o funcionalidades que los desarrolladores pueden usar.
- Permite comunicación entre dos aplicaciones distintas.
  - Con el sistema operativo (Cuadro de diálogo en Windows, Lector de huellas digitales en Android).
- Permite comunicación entre servicios.
  - Con Facebook (Autenticación, Comentarios, etc).
  - Google Maps (Sino tendríamos que crear nuestro propio sistema de mapas).
- Aunque técnicamente una API es un contrato a implementar, se lo utiliza como sinónimo de library (biblioteca) o servicio.
  - Library: Paquete de código reutilizable.
  - Servicio: Un sistema en ejecución que provee funcionalidad a otros sistemas y aplicaciones.

Terminología: Framework

- Se traduce como “Marco de trabajo”.
- Es una colección de APIs que facilitan el desarrollo y despliegue de aplicaciones que están basadas en una tecnología en particular.
  - Como un esqueleto o esquema sobre el cual construir aplicaciones específicas de forma más eficiente y segura.
- Las implementaciones de .NET no son frameworks, aunque con frecuencia se los menciona como tales (incluso en la documentación oficial).
- En la cátedra vamos a ver Windows Forms, el cual es un framework para crear aplicaciones de escritorio utilizando la API gráfica de Windows.
  - Soportado por las implementaciones .NET Framework y .NET Core v3.0.

Framework vs Library

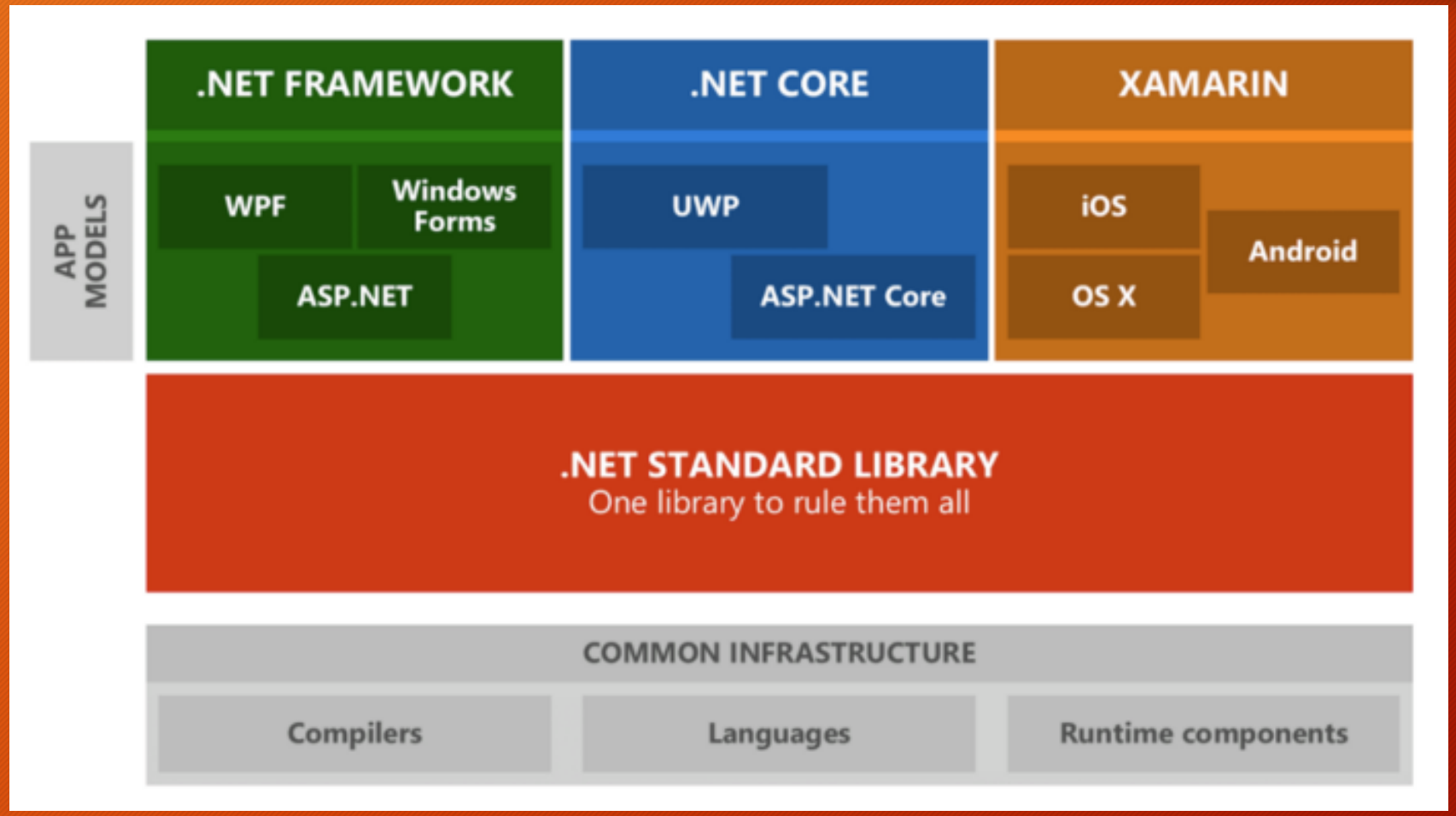
Framework	Library
Inversión de dependencia: El Framework llama y ejecuta el código generado por nosotros.	Nosotros llamamos a las funciones de la biblioteca.



Implementaciones de .NET

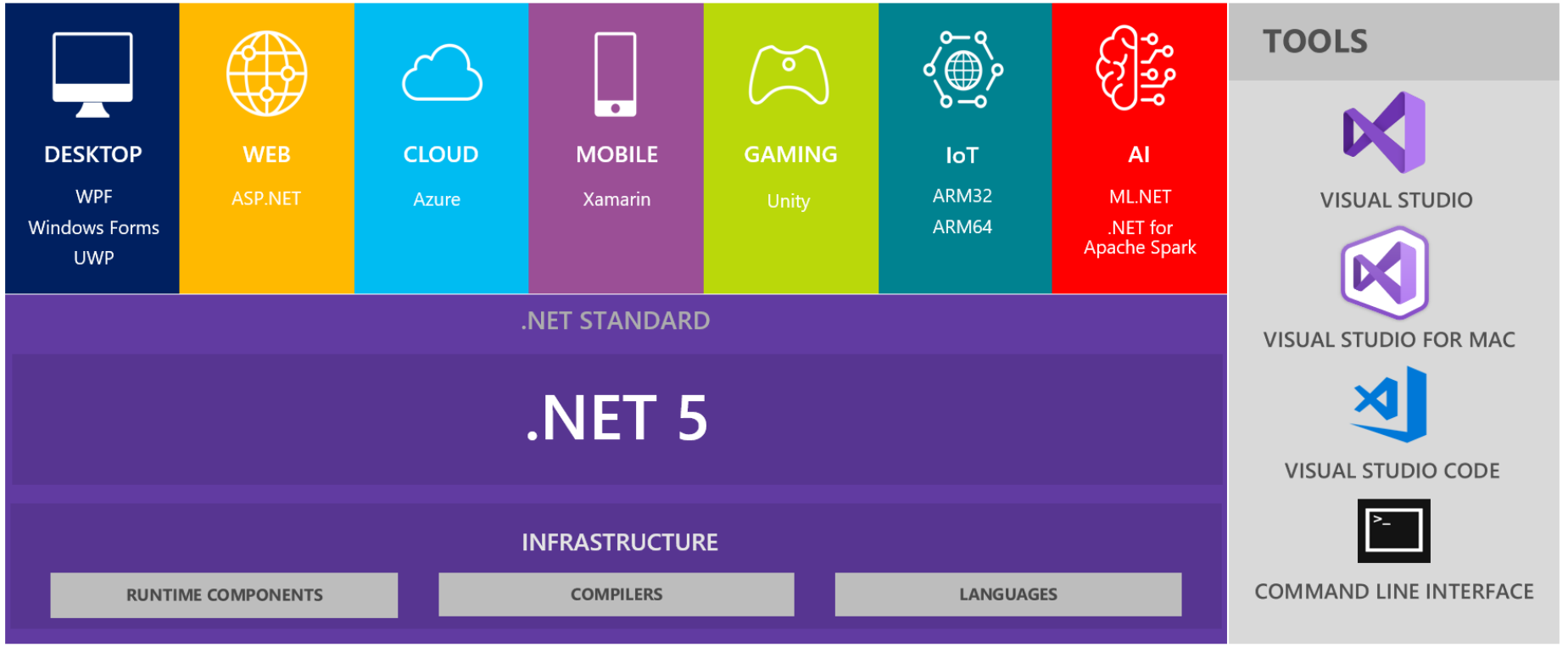
- Incluyen:
  - Uno o más entornos de ejecución (runtime).
    - CLR para .NET Framework.
    - CoreCLR para .NET Core.
  - Una biblioteca de clases que implementa .NET Standard y puede implementar APIs adicionales.
    - .NET Framework Base Class Library.
    - .NET Core Base Class Library.
  - Opcionalmente, uno o más frameworks.
    - ASP.NET (Web)
    - Windows Forms (Desktop)
    - Windows Presentation Foundation (WPF) (Desktop)
  - Opcionalmente, herramientas de desarrollo (Ej. Visual Studio).

.NET Stack


.NET FRAMEWORK
.NET CORE
XAMARIN
APP MODELS
WPF
Windows Forms
ASP.NET
UWP
ASP.NET Core
iOS
Android
OS X
.NET STANDARD LIBRARY
One library to rule them all
COMMON INFRASTRUCTURE
Compilers
Languages
Runtime components

Lo que se viene...

.NET – A unified platform



CTS – Common Type System

- Es un conjunto común de TIPOS DE DATOS y reglas que cualquier lenguaje de .NET Framework debe implementar.
- Provee una biblioteca que contiene los tipos primitivos básicos.
- Establece un marco que permite la ejecución inter-lenguajes.

Tipos de Datos

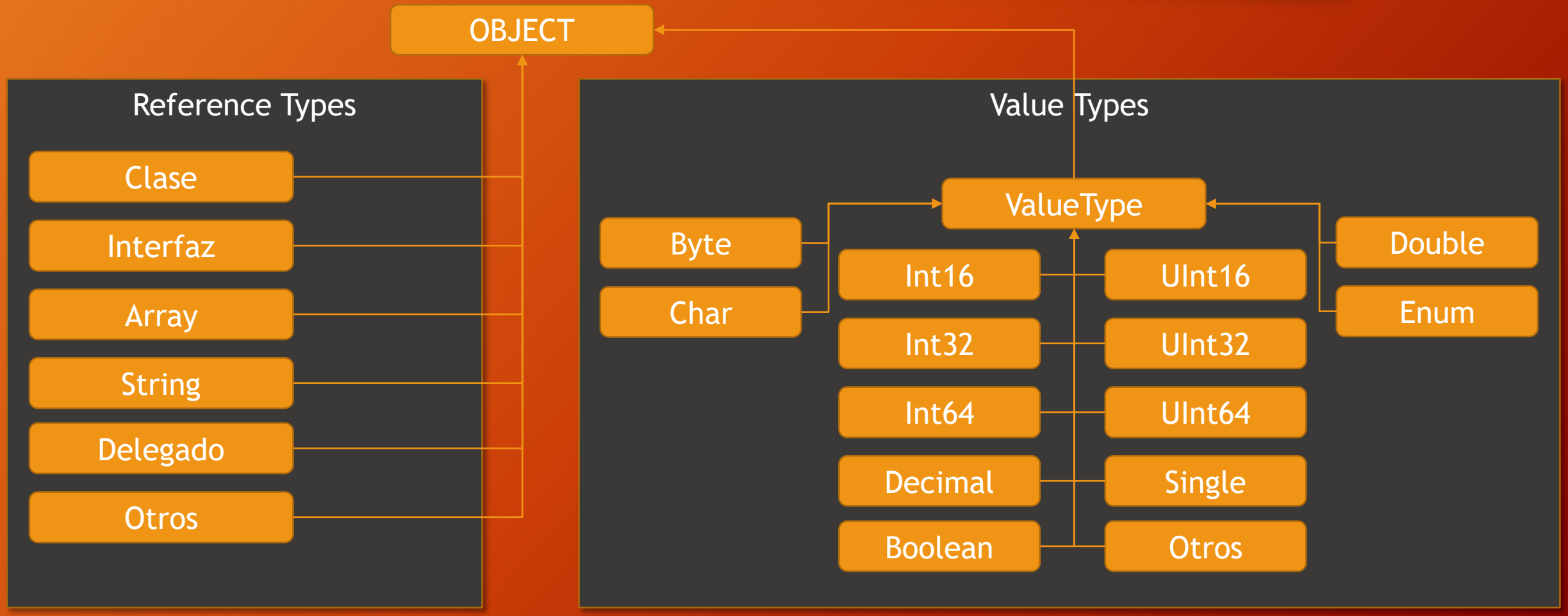
- **Reference Types:**
  - Estos tipos son representados por una referencia a la ubicación en memoria del valor actual del objeto, similar a un puntero en C.
  - Si asignan un Reference Type a una variable y luego la pasan a una función, cualquier cambio en el objeto se verá reflejado. NO se genera una copia.

- **Value Types:**
  - Estos tipos son representados por sus valores.
  - Si asignan un Value Type a una variable se copia el valor.

- **Variables Escalares:**
  - Las variables escalares son constantes o variables que contienen un dato atómico y unidimensional.

- **Variables No Escalares:**
  - Las variables no escalares son array (vector), lista y objeto, que pueden tener almacenado en su estructura más de un valor.

Common Type System (CTS)



Categoría	Clase	Descripción	C# Alias
Enteros	Byte	Un entero sin signo (8-bit)	byte
	SByte	Un entero con signo (8-bit)	sbyte
	Int16	Un entero con signo (16-bit)	short
	Int32	Un entero con signo (32-bit)	int
	Int64	Un entero con signo (64-bit)	long
Punto Flotante	Single	Un número de punto flotante de simple precisión (32-bit)	float
	Double	Un número de punto flotante de doble precisión (64-bit)	double
	Decimal	Un número decimal de 96-bit	decimal
Lógicos	Boolean	Un valor booleano (true o false)	bool
Otros	Char	Un caracter Unicode (16-bit)	char
	Object	La raíz de la jerarquía de objetos	object
	String	Una cadena de caracteres unicode inmutable y de tamaño fijo	string

Valores Predeterminados

- Enteros
  - 0 (cero)
- Punto flotante
  - 0 (cero)
- Lógicos
  - False
- Referencias
  - Null

Conversiones Básicas

- **Implícitas: no interviene el programador**

```
float flotante = 15;
```

- **Explícitas: interviene el programador, ya que puede haber perdida de datos.**

```
int entero = (int)15.2;
```

Conversiones Implícitas

Tipo	Conversiones permitidas
sbyte	short, int, long, float, double, decimal
byte	short, ushort, int, uint, long, ulong, float, double, decimal
short	int, long, float, double, decimal
ushort	int, uint, long, ulong, float, double, decimal
int	long, float, double, decimal
uint	long, ulong, float, double, decimal
long	float, double, decimal
ulong	float, double, decimal
float	double
double	Ninguna
decimal	Ninguna

Operadores Aritméticos

Descripción	C#
Asignación	=
Adición	+
Sustracción	-
Multiplicación	*
División	/
Negación	!
Módulo (Parte entera de la división)	%
Mayor	>
Menor	<
Mayor o Igual	>=
Menor o Igual	<=

Operadores Lógicos

Descripción	C#
Operador lógico Y	&&
Operador lógico O	||
Negación lógica	!
Igualdad	==
Desigualdad	!=

En C# todas las evaluaciones se hacen por “cortocircuito”

```
//Si Hacer1() es True,
//entonces NO se evalúa Hacer2()
if (Hacer1() || Hacer2())
{ }
```

```
//Si Hacer1() es False,
//entonces NO se evalúa Hacer2()
if (Hacer1() && Hacer2())
{ }
```

Sentencias Condicionales

```
if (x > 10)
    Hacer1();
```

```
if (x > 10)
{
    Hacer1();
    Hacer2();
}
```

```
if (x > 10)
{
    Hacer1();
}
else
{
    Hacer2();
}
```

```
if (x > 10)
{
    Hacer1();
}
else if (x <= 20)
{
    Hacer2();
}
else
{
    Hacer3();
}
```

Sentencias Condicionales

```
int a = 0;

switch (a)
{
    case 1:
        // Código 1
        break;
    case 2:
        // Código 2
        break;
    default:
        // Código DEFAULT
        break;
}
```

Sentencias Repetitivas

```
// Partes: declaración, prueba, acción
for (int i = 1; i < 10; i++)
{
}
```

La sentencia **foreach** permite recorrer arreglos y colecciones

```
string[] nombres = new string[5];

foreach (string auxNombre in nombres)
{
    //auxNombre es un elemento de nombres.
}
```

Sentencias Repetitivas

```
bool condicion = true;

while (condicion == true)
{
    //En algún momento poner condicion = false
}
```

```
bool condicion = true;

do
{
    //En algún momento poner condicion = false
} while (condicion == true);
```

Entry Point

```
class HolaMundo
{
    static void Main(string[] args)
    {
        System.Console.WriteLine("Hola mundo C#");
        // nombre completamente cualificado.

        System.Console.ReadKey();
    }
}
```

Entry Point

El punto de entrada para los programas en C# es la función Main

- static: Es un modificador que permite ejecutar un método sin tener que instanciar a una variable (sin crear un objeto). El método Main() debe ser estático.
- void: Indica el tipo de valor de retorno del método Main(). No necesariamente tiene que ser void.
- string [] args: Es un Array de tipo string que puede recibir el método Main() como parámetro. Este parámetro es opcional.

Console

- Es una clase pública y estática.
- Representa la entrada, salida y errores de Streams para aplicaciones de consola.
- Es miembro del NameSpace **System**.

Console: Métodos

- Clear()
  - Limpia el buffer de la consola. Equivalente a **clrscr()** de C.
- Read()
  - Lee el próximo carácter del stream de entrada. Devuelve un entero.
- ReadKey(bool)
  - Obtiene el carácter presionado por el usuario. La tecla presionada puede mostrarse en la consola. Equivalente a **getch()** / **getche()** de C.

Console: Métodos

- ReadLine()
  - Lee la siguiente línea de caracteres de la consola. Devuelve un string. Equivalente a **gets()** de C.
- Write()
  - Escribe el string que se le pasa como parámetro a la salida estándar. Equivalente a **printf()** de C.
- WriteLine()
  - Ídem método Write, pero introduce un salto de línea al final de la cadena.

Console: Propiedades

- BackGroundColor: Obtiene o establece el color de fondo de la consola.
- ForeGroundColor: Obtiene o establece el color del texto de la consola.
- Title: Obtiene o establece el título de la consola.

Formato de salida de Texto

Con los marcadores (“{}”), además de indicar el número de parámetro que se usará, podemos indicar la forma en que se mostrará.

Cuantos caracteres se mostrarán y si se formatearán a la derecha o la izquierda o también se pueden indicar otros valores de formato.

```
string miTexto = "Hola!";


Console.Write("Acá iría mi texto: {0}", miTexto);
```

Formato de salida de Texto

Fomato completo: { N [, M][: Formato] } (*)

- **N** será el número del parámetro, empezando por cero.
- **M** será el ancho usado para mostrar el parámetro, el cual se rellenará con espacios. Si M es negativo, se justificará a la izquierda, y si es positivo, se justificará a la derecha.
- **Formato** será una cadena que indicará un formato extra a usar con ese parámetro.

Formato de salida de Texto

```
Console.WriteLine("{0,10} {1,-10}{2}", 10, 15, 23);
// Salida:
//          10 15          23

Console.WriteLine("{ 0,10:0.00}{1,10}", 10.476, 15.355);
// Salida:
//          10.48          15.355
```

Formatos de Fecha

Character	Description	Usage	Example
d	Short Date	{0:d}	29-05-2018
D	Long Date	{0:D}	29 May 2018
t	Short Time	{0:t}	05:29:20
T	Long Time	{0:T}	05:29:20
f or F	Long Date Time	{0:f}	29 May 2018 05:30:08
g or G	Short Date Time	{0:g}	29-05-2018 05:31:42
M	Short Date	{0:M}	May 29
r	RFC1123 Date Time String	{0:r}	Tue, 29 May 2018 05:33:02 GMT
s	Sortable Date/Time	{0:s}	2018-05-29T05:34:10
u	Universal Sortable Date	{0:u}	2018-05-29 05:35:47Z
U	Universal full date	{0:U}	29 May 2018 00:08:07
Y	Year month pattern	{0:Y}	May, 2018

Formatos Numéricos

Character	Description	Usage	Example
c	Currency	{0:c}	$ 75,674.74
e	Scientific	{0:e}	7.567474e+004
f	Fixed Point	{0:f}	75674.74
g	General	{0:g}	75674.73789621
n	Thousand Separator	{0:n}	75,674.74

Formatos Personalizables

Character	Description	Usage	Example
0	Zero Placeholder	{0:00.00}	75674.74
#	Digit Placeholder	{0:(#).##}	(75674).74
.	Decimal Point	{0:0.000}	75674.738
,	Thousand Separator	{0:0,0}	75,675
%	Percent	{0:0%}	7567474%

Bibliografía

- **.NET architectural components:** https://docs.microsoft.com/en-us/dotnet/standard/components
- **Glosario .NET:** https://docs.microsoft.com/en-us/dotnet/standard/glossary