FUNDAMENTOS C#

```
1  // Programa sencillo C# para mostrar el uso de los identificadores
2  using System;
3
4  class AprendiendoCSharp { // 1
5
6      // Método Main
7      static public void Main() // 2
8      {
9
10         // variable
11         int numberOne = 10; // 3
12         int numberTwo = 39; // 4
13         int result;         // 5
14
15         // Una simple suma de números
16         result = numberOne + numberTwo;
17         Console.WriteLine("La suma de los dos números es: {0}",
           result);
18     }
19 }
```

Master en C# con Visual Studio Code

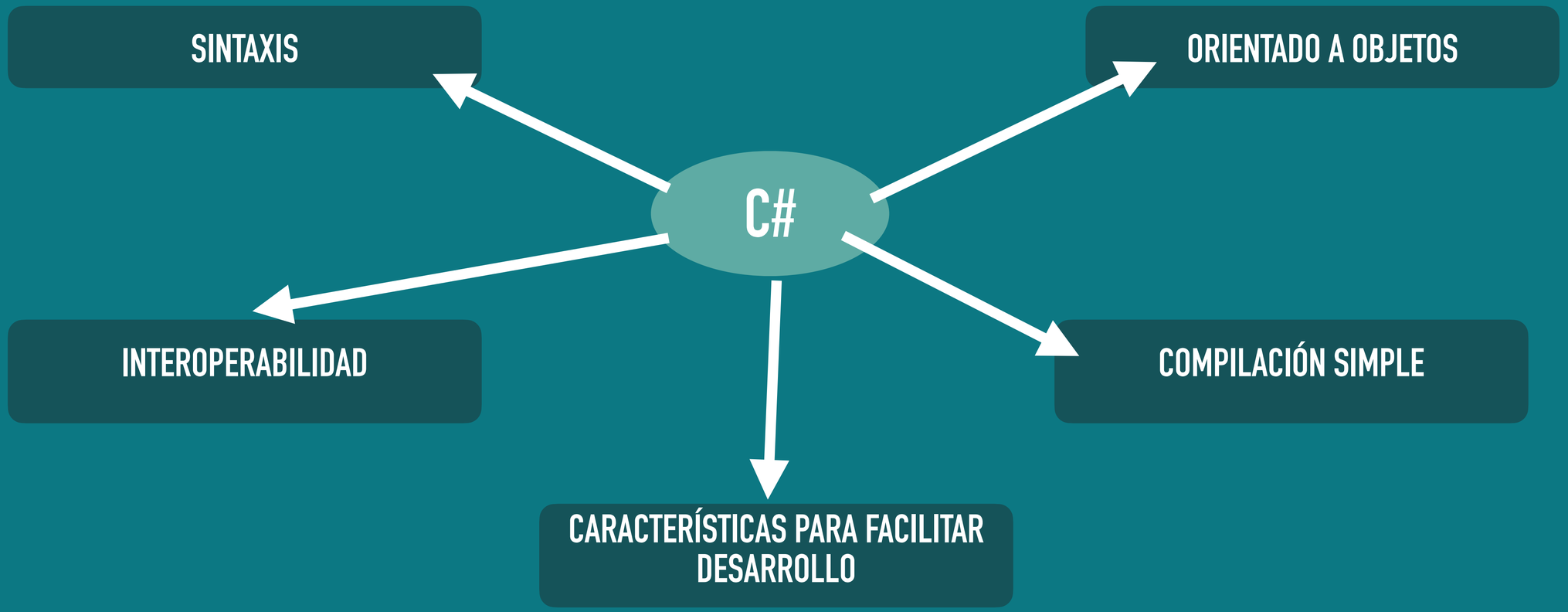
¿QUÉ ES C#?

- Lenguaje de programación multiparadigma.
- Desarrollado y estandarizado por Microsoft como parte de su plataforma .NET.
- Su sintaxis básica deriva de C / C++.
- Utiliza modelo objetos de .Net, similar a Java, con mejoras.
- Aunque forma parte de .NET, que es una API, es un lenguaje de programación independiente.

¿CÓMO ESTARÁ ENFOCADA ESTA SECCIÓN?

- Características de C#.
- Diferencias C# / .NET.
- Arquitectura de aplicaciones .NET.
- Instalaciones
- Primer programa en C#.
- Identificadores y reglas de definición dentro del programa.
- Keywords (Palabras clave).
- Convención de nombres (name conventions).
- Comentarios.

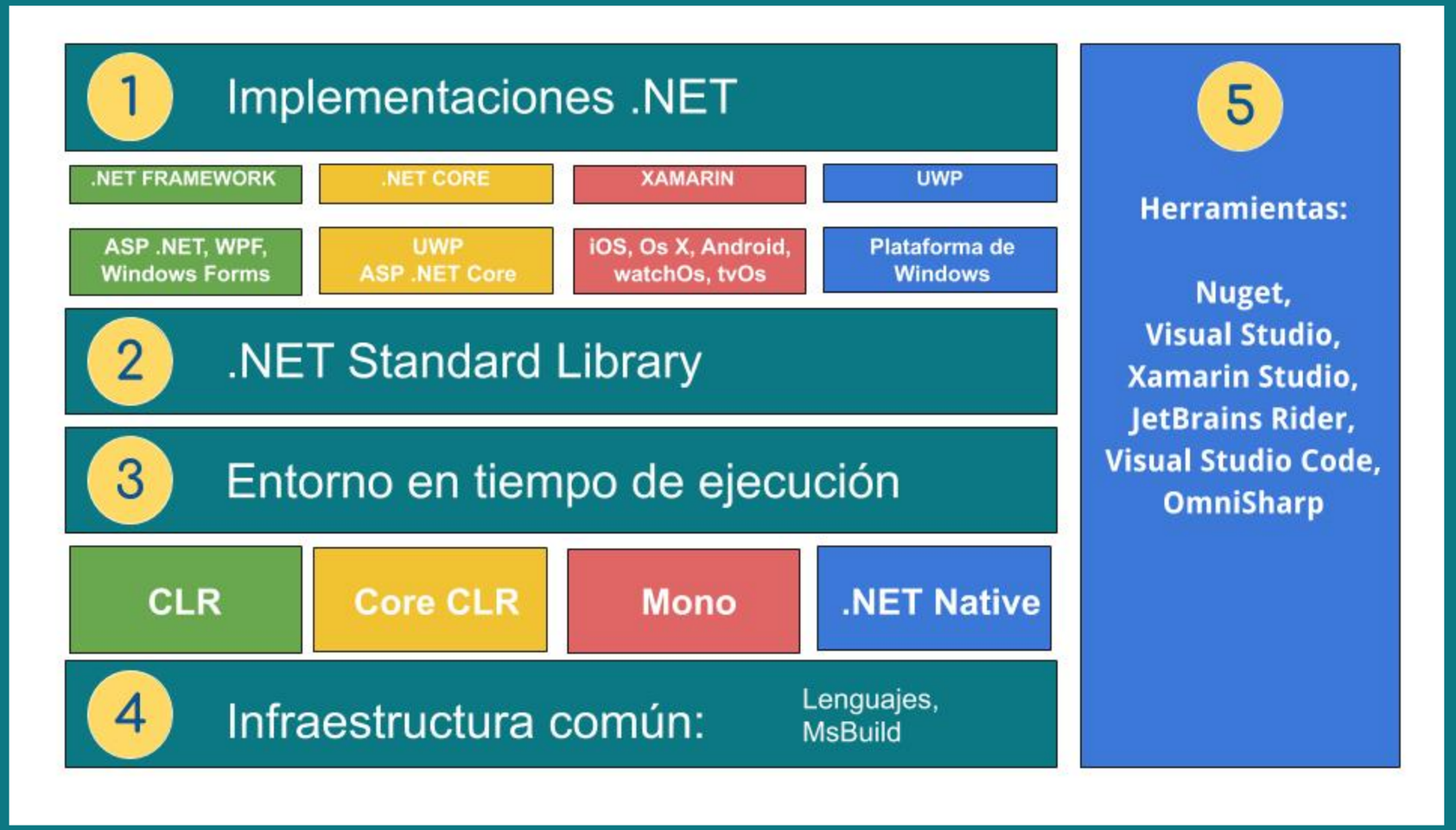
CARACTERÍSTICAS PRINCIPALES



DIFERENCIAS C# Y .NET

- C# es un lenguaje de programación / .NET no.
- Implementación de C# muy integrada con .NET Framework.
- .NET no solo biblioteca, también tiempo de ejecución para ejecutar aplicaciones.
- El conocimiento de C# implica conocimiento de .NET.
- Se pueden usar otros lenguajes como F#, VB .NET y C++ en .NET

COMPONENTES ARQUITECTURA DE APLICACIONES .NET



Lo que vamos a usar:

1. .NET CORE
2. .NET Standard Library
3. Core CLR
4. C# + MsBuild
5. Visual Studio Code + Omnisharp

INSTALACIONES NECESARIAS

- Descargar Visual Studio Code: https://code.visualstudio.com/
- Extensiones en Visual Studio Code: https://marketplace.visualstudio.com/vscode
- Instalar Dotnet para trabajar con .Net Core: https://dotnet.microsoft.com/download

Referencias a tener en cuenta:

https://docs.microsoft.com/es-es/dotnet/csharp/tutorials/

https://docs.microsoft.com/es-es/dotnet/ (Documentación oficial)

IDENTIFICADORES – DEFINICIÓN Y CONCEPTOS

- Nombre que asigna él programador a elementos del programa.
- Formado por letras y dígitos.
- Objetivo => identificar los diferentes componentes del programa.

```
public class AprendiendoCSharp {
    static public void Main ()
    {
        int x;
    }
}
```

IDENTIFICADORES – REGLAS DE DEFINICIÓN DENTRO DEL PROGRAMA

- Hay que seguir unas reglas para definir un identificador.
- Si no se cumple alguna de ellas => ERROR DE COMPILACIÓN

Carácteres alfanuméricos [A-Z], [a-z], [0-9], “_”	valueInput, firstNumber, Coins, … “anartz @“
No comenzar con dígitos	123Anartz, 09Values
No espacios	“Anartz Mugika”
No permitido ninguna palabra clave (@ antes para ser válido)	as / @as
Distinguen mayúsculas / minúsculas	Anartz <> anartz
No + de 512 caracteres	
No añadir dos guiones bajos. Se usa para implementaciones	Anartz_Mugika / A__M

IDENTIFICADORES – EJEMPLO SENCILLO

```
1  // Programa sencillo C# para mostrar el uso de los identificadores
2  using System;
3
4  class AprendiendoCSharp { // 1
5
6      // Método Main
7      static public void Main() // 2
8      {
9
10         // variable
11         int numberOne = 10; // 3
12         int numberTwo = 39; // 4
13         int result;         // 5
14
15         // Una simple suma de números
16         result = numberOne + numberTwo;
17         Console.WriteLine("La suma de los dos números es: {0}", result);
18     }
19 }
```

KEYWORDS (PALABRAS CLAVE)

- Palabras que se usan en un lenguaje para un proceso interno o acciones predefinidas.
- No pueden usarse como nombres de variables u objetos.
- Existen 78 palabras clave.
- Existe la opción de usar estas palabras clave si añadimos delante “@“. Por ejemplo la palabra reservada “if”, “@if”

KEYWORDS (PALABRAS CLAVE)

abstract	decimal	float	namespace	return	try
as	default	for	new	sbyte	typeof
base	delegate	foreach	null	sealed	unit
bool	do	goto	object	short	ulong
break	double	if	operator	sizeof	unchecked
byte	else	implicit	out	stackalloc	unsafe
case	enum	in	override	static	ushort
catch	event	int	params	string	using
char	explicit	interface	private	struct	using static
checked	extern	internal	protected	switch	virtual
class	FALSE	is	public	this	void
const	finally	lock	readonly	throw	volatile
continue	fixed	long	ref	TRUE	while

KEYWORDS (PALABRAS CLAVE) – EJEMPLO CÓDIGO

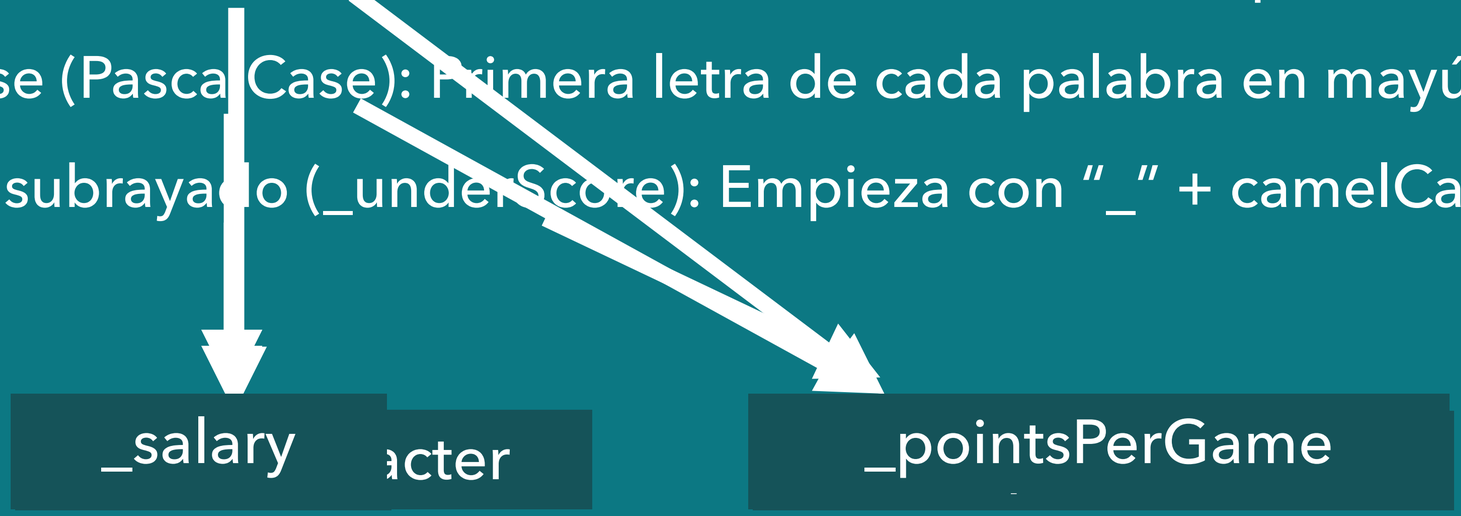
```
1  // Programa sencillo C# para mostrar el uso de los identificadores
2  using System;
3
4  class AprendiendoCSharp { // 1
5
6      // Método Main
7      static public void Main() // 2
8      {
9
10         // variable
11         int numberOne = 10; // 3
12         int numberTwo = 39; // 4
13         int result;         // 5
14
15         // Una simple suma de números
16         result = numberOne + numberTwo;
17         Console.WriteLine("La suma de los dos números es: {0}", result);
18     }
19 }
```

Keywords en programa:

- using
- class
- static
- public
- void
- int

CONVENCIÓN DE NOMBRES

- Son las normas usadas para aplicar unas buenas prácticas comunes.
- Hay 3 maneras diferentes de declarar estándares de nomenclatura de C# / .Net:
  - Camel Case (camelCase): Primera letra minúscula. Cada palabra en mayúsculas
  - Pascal Case (PascalCase): Primera letra de cada palabra en mayúsculas.
  - Prefijo de subrayado (_underScore): Empieza con “_” + camelCase



CONVENCIÓN DE NOMBRES

Tipo	Regla
Private (propiedad)	_lowerCamelCase
Public (propiedad)	UpperCamelCase
Protected (propiedad)	UpperCamelCase
Internal (propiedad)	UpperCamelCase
Propiedad	UpperCamelCase
Método / Función	UpperCamelCase
Clase (Class)	UpperCamelCase
Interface	IUpperCamelCase
Variable local	lowerCamelCase
Argumentos / Parámetros	lowerCamelCase

```
1  using System;
2
3  class MainClass {// <==== PascalCase = Class / Clase
4    // Propiedades
5    private int _salary = 100;  // Private 1
6    public int Salary          // Property
7    {
8      get      2
9      {
10         return _salary;
11     }
12     set
13     {
14         _salary = value;
15     }
16   }
17   // Método / Función Máximo 7 parámetros y PascalCase = GetPosts
18   public string GetPosts(string postId) 4
19   {
20     // Argumento / Parámetro ==> postId (camelCase)
21   5 int numberPost = 1; // <=== local variable (camelCase)
22     return Convert.ToString(numberPost);
23   }
24
25
26   public static void Main (string[] args) {
27     Console.WriteLine ("Name conventions");
28   }
29 }
```

COMENTARIOS

- Sirven para explicar el código de C# y hacerlo + legible.
- También para evitar ejecutar un código cuando usamos código alternativo.
- Hay varios tipos:
  - Una línea.
  - Dos líneas.
  - Documentación del código con comentarios XML.

COMENTARIOS – UNA LÍNEA

```
// Mensaje de bienvenida que se imprimirá en la consola
Console.WriteLine ("Hello World");
```

```
Console.WriteLine ("Hello World"); // Comentario de una línea
```

COMENTARIOS – MULTILÍNEA

```
/**
Preparado para trabajar con comentario de varias líneas.
**/
Console.Write("Mensaje después de comentario multilínea");
```

```
/**
Este comentario será de varias líneas donde
daremos una explicación más extensa con más detalles.
Nombre: Creando comentarios
Autor:
**/
Console.Write("Mensaje después de comentario multilínea");
```