**GUÍA DE EXPRESIONES LAMBDA EN C#**

**Objetivo general**

Comprender y aplicar expresiones lambda en C# como una forma concisa y funcional de definir funciones anónimas, reconociendo su sintaxis, estructura, tipos asociados (Func<> y Action<>) y sus usos fundamentales.

**Objetivos específicos**

* Identificar la sintaxis básica de una expresión lambda en C#.
* Diferenciar las expresiones lambda de los métodos tradicionales.
* Comprender el uso de los tipos genéricos Func<> y Action<>, especificando correctamente los tipos de parámetros y de retorno.
* Reconocer que el último tipo en Func<> corresponde al valor que devuelve la función.
* Aplicar expresiones lambda con diferentes cantidades y tipos de parámetros, usando tanto expresiones simples como bloques de código.
* Establecer las diferencias entre Func<> y Action<> según la necesidad de retorno de valor.
* Utilizar expresiones lambda en contextos básicos sin depender de LINQ, APIs ni bibliotecas externas.
* Preparar el terreno conceptual para avanzar posteriormente a temas como delegados personalizados, closures, o expresiones lambda en LINQ.

**1. ¿Qué es una expresión lambda?**

Una expresión lambda es una forma abreviada de declarar una función anónima. Es decir, una función sin nombre que se puede almacenar en una variable, pasar como argumento o usar directamente.

**Forma general:**

(parámetros) => expresión o bloque

**2. ¿Para qué sirve?**

Sirve para definir lógica en **una sola línea o bloque**, sin necesidad de crear un método completo.

**3. Tipos base: Func y Action**

Func<...>: devuelve un valor

* Su sintaxis es:

Func<tipo1, tipo2, ..., tipoResultado>

El último tipo SIEMPRE es el tipo del resultado devuelto por la lambda.

Ejemplo:

Func<int, int, int> suma = (a, b) => a + b;

* int: primer parámetro (a)
* int: segundo parámetro (b)
* int: tipo del valor que devuelve la función

**Action<...>: no devuelve valor (void)**

Ejemplo:

Action<string> saludar = nombre => Console.WriteLine($"Hola, {nombre}");

**4. Ejemplos de menor a mayor**

**Lambda sin parámetros, que devuelve un valor:**

Func<string> obtenerMensaje = () => "Hola mundo";

**Lambda con un parámetro:**

Func<int, int> cuadrado = x => x \* x;

**Lambda con dos parámetros:**

Func<int, int, int> multiplicar = (x, y) => x \* y;

**Lambda con parámetros de tipos diferentes:**

Func<string, int, string> repetir = (texto, veces) => texto + veces.ToString();

**Lambda con bloque de código:**

Cuando hay más de una instrucción, se usan llaves {}.

Func<int, int, int> dobleSuma = (x, y) =>

{

int suma = x + y;

return suma \* 2;

};

**5. Reglas clave**

* Los nombres de los parámetros los eliges tú.
* Puedes usar paréntesis para agrupar parámetros: (x, y)
* Si hay un solo parámetro, los paréntesis son opcionales: x => x + 1
* El tipo de retorno debe coincidir con el declarado en Func<>

**Tabla 1.**

*Diferencia entre Func<> y Action<>*

| **Característica** | **Func<>** | **Action<>** |
| --- | --- | --- |
| Devuelve valor | Sí | No (retorna void) |
| Ejemplo retorno | int, string, etc. | void |
| Uso típico | Cálculos, transformaciones | Mostrar, escribir en consola |

**7. Diferencia entre lambda y método tradicional**

**Lambda:**

Func<int, int> cuadrado = x => x \* x;

**Método:**

int Cuadrado(int x) {

return x \* x;

}

**8. ¿Se pueden devolver tipos complejos?**

Sí. Aquí un ejemplo sin clases externas:

Func<string, int, string[]> separar = (palabra, veces) =>

{

return Enumerable.Repeat(palabra, veces).ToArray();

};

**Conclusión hasta aquí**

1. Una lambda es una función anónima.
2. Se puede guardar en una variable.
3. Func<> devuelve un valor → último parámetro es el tipo de retorno
4. Action<> no devuelve valor → útil para cosas como Console.WriteLine()
5. Los tipos de los parámetros pueden ser diferentes entre sí.
6. Las lambdas pueden tener una expresión o un bloque con llaves.

**EJERCICIOS DE EXPRESIONES LAMBDA EN C#**

**Ejercicio 1: Lambda sin parámetros**

**Enunciado:**  
Declara una expresión lambda que no reciba ningún parámetro y devuelva el texto "¡Bienvenido al curso de C#!".

**Instrucción:**

* Usa Func<string>
* Llama a la función y muestra el resultado por consola

**Ejercicio 2: Lambda con un parámetro**

**Enunciado:**  
Crea una lambda que reciba un número entero y devuelva su doble.

**Instrucción:**

* Usa Func<int, int>
* Pruébalo con el número 7

**Ejercicio 3: Lambda con dos parámetros**

**Enunciado:**  
Define una lambda que reciba dos números (int) y devuelva el **mayor** de los dos.

**Instrucción:**

* Usa Func<int, int, int>
* Pruébalo con los valores 12 y 9

**Ejercicio 4: Lambda con tipos mezclados**

**Enunciado:**  
Crea una lambda que reciba un string (nombre) y un int (edad), y devuelva una frase como:

"Juan tiene 25 años"

**Instrucción:**

* Usa Func<string, int, string>
* Pruébalo con "Juan" y 25

**Ejercicio 5: Lambda con bloque de código**

**Enunciado:**  
Declara una lambda que reciba tres enteros y devuelva el promedio entero.

**Instrucción:**

* Usa Func<int, int, int, int>
* Usa llaves {} porque tendrás más de una línea
* Prueba con los valores 8, 9 y 10

**Ejercicio 6: Action sin retorno**

**Enunciado:**  
Crea un Action<string> que reciba un nombre y muestre en consola:

"Hola, [nombre]. ¡Qué bueno verte!"

**Ejercicio 7: Detectar error de tipo (teórico)**

**Pregunta:**  
¿Por qué este código da error?

Func<int, double, string> error = (a, b) => a + b;

Explica qué tipo está devolviendo la expresión, qué tipo espera el Func<>, y cómo lo corregirías.