**1️⃣ Principio central en programación funcional: funciones puras**

En programación funcional se busca que las funciones:

* Sean **deterministas** (mismo input → mismo output a través del tiempo).
* Sean **sin efectos colaterales** (no cambien nada fuera de ellas, ni variables globales, ni objetos externos, etc.).

⚠️ El problema es que, si trabajás con datos **mutables** (que se pueden cambiar), es muy fácil romper esa pureza sin querer:

// Impura: cambia algo de afuera

void CambiarNombre(Beer b)

{

b.Name = "Heineken"; // afecta al objeto externo

}

Por eso, en programación funcional **se favorece trabajar con datos inmutables**.

**2️⃣ Principio: inmutabilidad**

La inmutabilidad (como la de DateTime o string en C#):

* Evita que puedas modificar accidentalmente un valor fuera de la función.
* Hace que sea más fácil cumplir el requisito de “**sin efectos colaterales**”.

En el ejemplo del video:

* DateTime es **struct** (pasa por valor) y **inmutable**.
* Esto significa que al pasarlo a la función, **no hay riesgo de que la función cambie el original**.
* Eso encaja perfecto con la mentalidad funcional: *los datos entran, se procesan, y sale un nuevo dato; el original queda intacto*.

**3️⃣ Conexión entre todo**

* **Programación funcional** → busca funciones puras + datos inmutables.
* **Struct inmutable (DateTime)** → es un tipo de dato que encaja perfecto porque no rompe pureza.

💡 **Idea clave**:

En programación funcional, la inmutabilidad no es un capricho: es una forma práctica de garantizar que las funciones sean puras y que el estado no se altere de forma inesperada.

**Existen 2 Tipos de Funciones:**

1) **Funciones de primera clase**

En programación funcional decimos que las funciones son de primera clase cuando se pueden tratar como cualquier otro dato:

* Guardarlas en variables.
* Pasarlas como parámetros.
* Devolverlas desde otras funciones.

**2) Funciones de orden superior**

Una **función de orden superior** es aquella que:

1. **Recibe** otra función como parámetro.
2. **O devuelve** una función como resultado.
3. (O ambas cosas)

Esto te permite que el comportamiento de la función sea más **flexible y configurable**.

**Delegados en C#**

En C#, un delegado es un tipo que representa la referencia a un método con una firma específica.

En vez de definir delegados a mano, normalmente usamos:

* Action → representa una función que no devuelve nada (void).
* Func → representa una función que devuelve algo.