## **El Principio DRY: No te Repitas (Don't Repeat Yourself)**

El principio DRY es un concepto fundamental en la programación que busca **evitar la duplicación de código**. En lugar de repetir el mismo código en diferentes partes de un programa, se busca encapsular esa lógica en una sola ubicación y reutilizarla cuando sea necesario. Esto no solo hace que el código sea más **conciso y legible**, sino que también facilita su **mantenimiento**, ya que cualquier cambio que deba hacerse solo necesita realizarse en un solo lugar.

### **¿Por qué es importante el principio DRY?**

* **Mayor legibilidad:** Código más fácil de entender y seguir.
* **Menor probabilidad de errores:** Al reducir la cantidad de código, se disminuyen las oportunidades de introducir errores.
* **Facilita el mantenimiento:** Los cambios se realizan en un solo lugar.
* **Reutilización de código:** Se promueve la creación de funciones y módulos reutilizables.

### **Implementando DRY en Python**

**1. Funciones:**

* **Encapsular lógica:** Crea funciones para realizar tareas específicas. Por ejemplo:
* Python
* def calcular\_area\_rectangulo(base, altura): return base \* altura
* Use code [with caution.](file:///C:/faq#coding)
* **Parametrizar:** Utiliza parámetros para hacer que las funciones sean más flexibles.
* Python
* def saludar(nombre): print(f"Hola, {nombre}!")
* Use code [with caution.](file:///C:/faq#coding)

**2. Clases:**

* **Abstraer conceptos:** Define clases para representar objetos del mundo real.
* **Métodos:** Crea métodos para encapsular el comportamiento de los objetos.
* Python
* class Perro: def ladrar(self): print("Guau!")
* Use code [with caution.](file:///C:/faq#coding)

**3. Módulos:**

* **Organizar código:** Divide el código en módulos para mejorar la estructura y la reutilización.
* Python
* # módulo calculadora.pydef sumar(a, b): return a + b def restar(a, b): return a - b
* Use code [with caution.](file:///C:/faq#coding)

# **otro módulo**

import calculadora

resultado = calculadora.sumar(3, 4)

\*\*4. Listas por comprensión y funciones lambda:\*\*

\* \*\*Crear listas de forma concisa:\*\*

```python

numeros = [1, 2, 3, 4]

cuadrados = [x\*\*2 for x in numeros]

* **Funciones anónimas:**
* Python
* lista = [1, 2, 3, 4, 5] resultado = list(map(lambda x: x\*2, lista))
* Use code [with caution.](file:///C:/faq#coding)

**5. Iteradores y generadores:**

* **Procesar datos de forma eficiente:**
* Python
* def generar\_numeros\_pares(n): for i in range(n): if i % 2 == 0: yield i
* Use code [with caution.](file:///C:/faq#coding)

### **Ejemplos Prácticos**

* **En lugar de:**
* Python
* print("El área del rectángulo es:", 5 \* 10) print("El área del rectángulo es:", 3 \* 7)
* Use code [with caution.](file:///C:/faq#coding)
* **Hacer:**
* Python
* def calcular\_area\_rectangulo(base, altura): return base \* altura print(calcular\_area\_rectangulo(5, 10)) print(calcular\_area\_rectangulo(3, 7))
* Use code [with caution.](file:///C:/faq#coding)
* **En lugar de:**
* Python
* if edad >= 18: print("Eres mayor de edad") else: print("Eres menor de edad")
* Use code [with caution.](file:///C:/faq#coding)
* **Hacer:**
* Python
* def verificar\_edad(edad): if edad >= 18: return "Eres mayor de edad" else: return "Eres menor de edad"
* Use code [with caution.](file:///C:/faq#coding)

### **Consideraciones Adicionales**

* **Abstracción:** Identifica los conceptos clave y crea abstracciones adecuadas (clases, funciones) para representarlos.
* **Reutilización:** Busca oportunidades para reutilizar código existente en lugar de duplicarlo.
* **Modularidad:** Divide el código en módulos para mejorar la organización y la mantenibilidad.
* **Nomenclatura clara:** Utiliza nombres descriptivos para funciones, variables y clases.

**Al aplicar el principio DRY, lograrás escribir código más limpio, eficiente y fácil de mantener a largo plazo.**