## **Principio de Responsabilidad Única (SRP)**

El principio SRP establece que una clase debe tener una única razón para cambiar. Es decir, una clase debe tener una única responsabilidad bien definida.

**¿Por qué es importante?**

* **Mantenibilidad:** Al tener clases con responsabilidades bien delimitadas, es más fácil modificar el código sin afectar otras partes del sistema.
* **Reutilización:** Clases con una sola responsabilidad son más fáciles de reutilizar en diferentes contextos.
* **Testabilidad:** Es más sencillo escribir pruebas unitarias para clases con una única responsabilidad.

**Ejemplo en Python: Clase Persona**

Imaginemos una clase Persona que, inicialmente, tiene estas responsabilidades:

Python

class Persona:

def \_\_init\_\_(self, nombre, apellido, edad):

self.nombre = nombre

self.apellido = apellido

self.edad = edad

def saludar(self):

print(f"Hola,

[1. github.com](https://github.com/arianafm/Python)

[github.com](https://github.com/arianafm/Python)

soy {self.nombre} {self.apellido}") def calcular\_impuestos(self, salario): # Lógica para calcular impuestos pass

Use code [with caution.](file:///C:/faq#coding)

Esta clase tiene dos responsabilidades:

1. **Representar a una persona:** Almacenar datos como nombre, apellido y edad.
2. **Calcular impuestos:** Realizar un cálculo relacionado con finanzas.

**Aplicando SRP:** Para cumplir con SRP, podemos dividir estas responsabilidades en dos clases:

Python

class Persona:

def \_\_init\_\_(self, nombre, apellido, edad):

self.nombre = nombre

self.apellido = apellido

self.edad = edad

def saludar(self):

print(f"Hola,

[1. github.com](https://github.com/arianafm/Python)

[github.com](https://github.com/arianafm/Python)

soy {self.nombre} {self.apellido}") class CalculadorDeImpuestos: def calcular\_impuestos(self, salario): # Lógica para calcular impuestos pass

Use code [with caution.](file:///C:/faq#coding)

Ahora, la clase Persona se encarga únicamente de representar a una persona, mientras que CalculadorDeImpuestos se enfoca en los cálculos de impuestos.

## **Detectando cuando un código no cumple con SRP**

Un código que viola SRP suele presentar los siguientes síntomas:

* **Métodos con nombres muy genéricos:** Métodos como hacer\_algo, procesar\_datos o manejar\_todo suelen indicar que la clase tiene múltiples responsabilidades.
* **Clases muy grandes:** Clases con muchas líneas de código y muchos métodos a menudo violan SRP.
* **Dificultad para entender el código:** Si es difícil entender qué hace una clase o un método, es probable que tenga demasiadas responsabilidades.
* **Cambios frecuentes en una misma clase:** Si una clase se modifica constantemente para agregar nuevas funcionalidades, es una señal de que podría estar violando SRP.

**Ejemplo de violación de SRP:**

Python

class CorreoElectronico: def enviar\_correo(self, destinatario, asunto, cuerpo): # Lógica para enviar el correo pass def formatear\_cuerpo(self, datos): # Lógica para formatear el cuerpo del correo pass

Use code [with caution.](file:///C:/faq#coding)

En este caso, la clase CorreoElectronico tiene dos responsabilidades: enviar el correo y formatear su cuerpo. Sería mejor separar estas responsabilidades en dos clases diferentes.

**En resumen:**

El principio SRP es fundamental para escribir código limpio, mantenible y escalable. Al aplicar SRP, tus clases serán más fáciles de entender, probar y modificar a lo largo del tiempo.