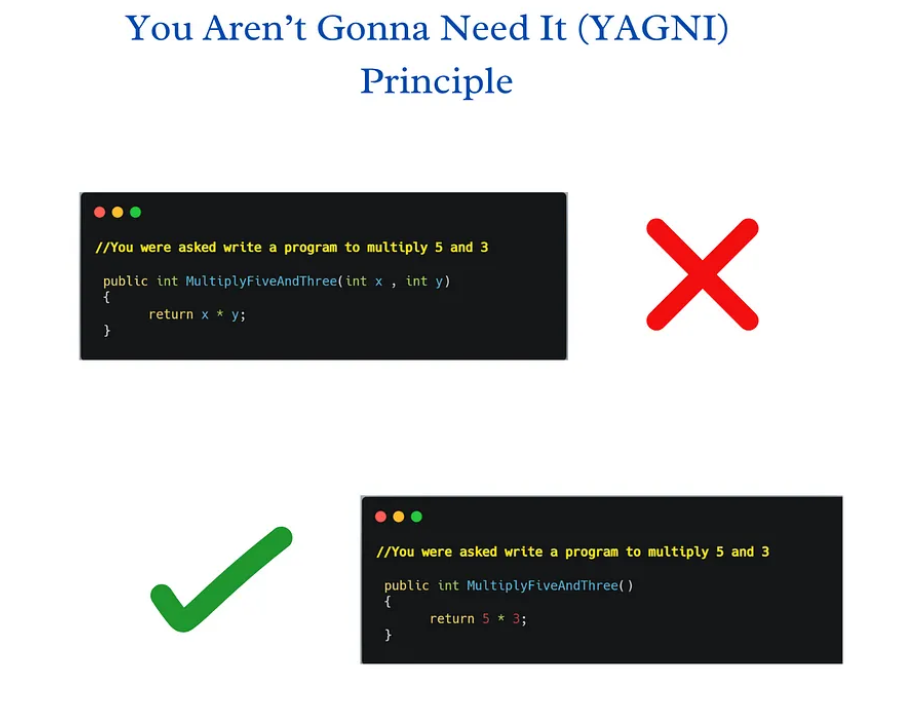
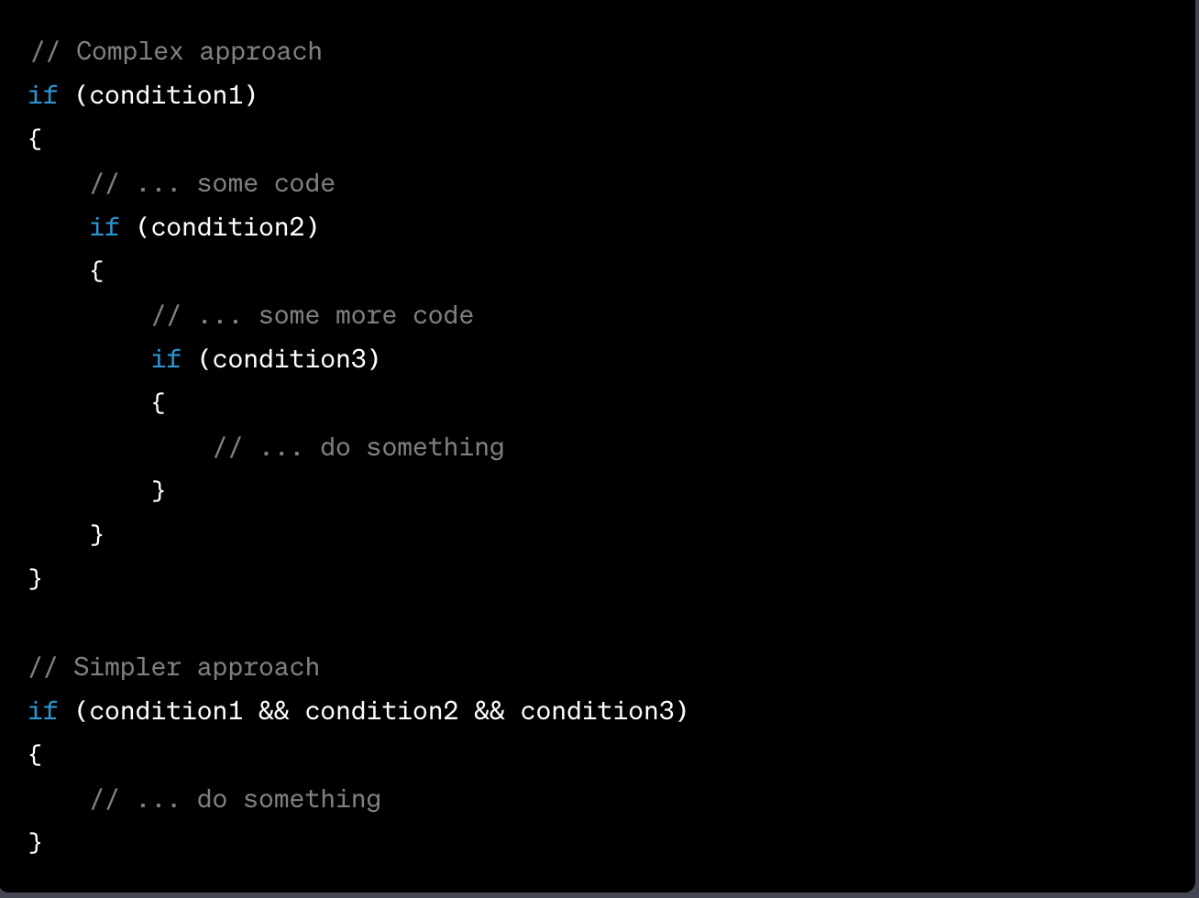
**PRINCIPIOS**

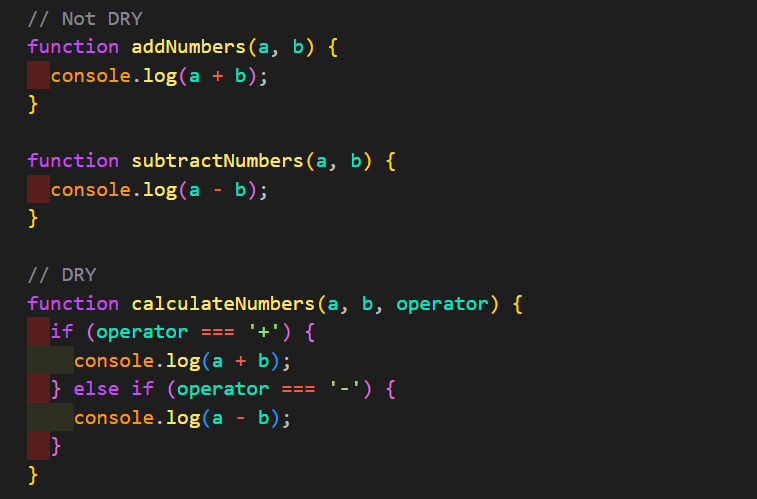
***YAGNI (You Aren't Gonna Need It):*** Este principio advierte contra la implementación de funcionalidades que no son necesarias en el momento presente. En lugar de agregar características o funcionalidades "por si acaso", se debe mantener el código lo más simple posible y agregar funcionalidades solo cuando sean necesarias. Esto ayuda a evitar la especulación e iterar desde proyectos iniciales dependiendo de las necesidades del usuario y si es requerido o no un elemento. En resumen, mantener lo esencial.

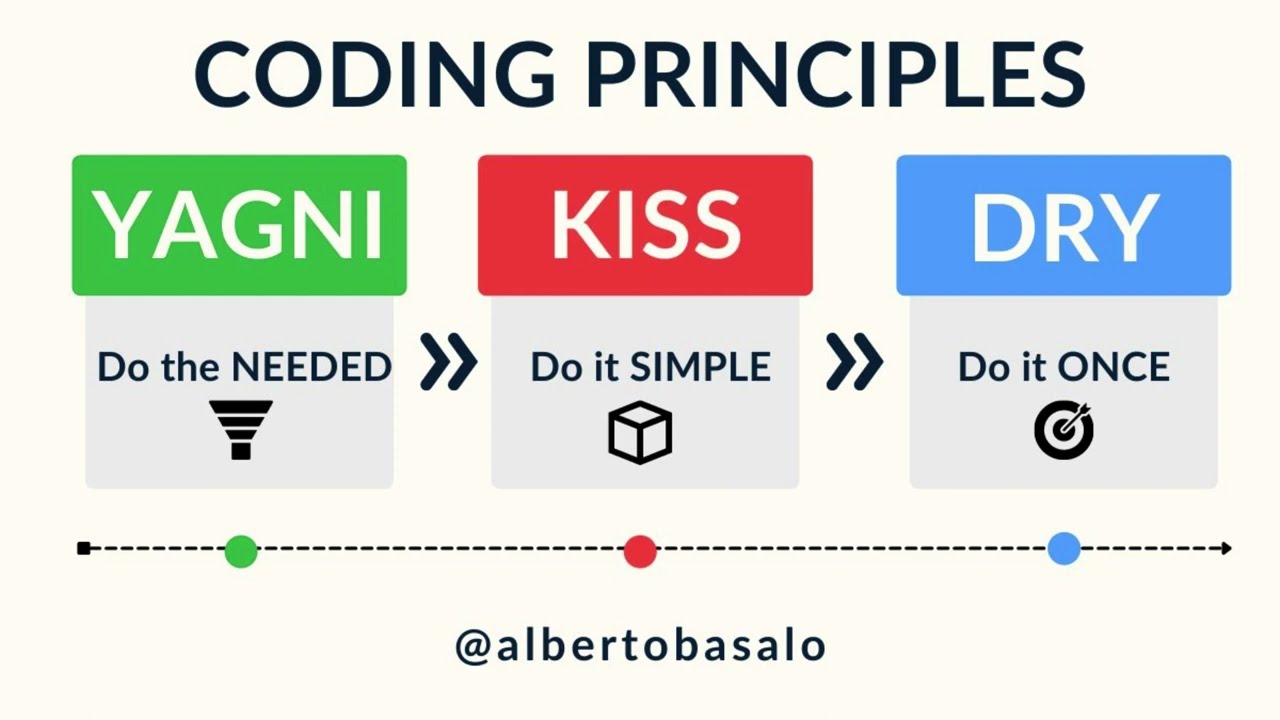


***KISS (Keep It Super Simple):*** este principio de diseño de software aboga por la simplicidad en el diseño y desarrollo. Se centra en mantener las soluciones simples y directas en lugar de complicarlas innecesariamente. Al seguir este principio, los desarrolladores pueden crear sistemas más fáciles de entender, mantener y escalar, lo que conduce a un código más robusto y menos propenso a errores.



***DRY (Don't Repeat Yourself):*** Este principio promueve la reutilización del código al evitar la repetición de lógica o datos similares en diferentes partes del sistema. En lugar de duplicar código, se debe extraer la lógica común a métodos o clases reutilizables. Esto hace que se focalice el código y se sepa dónde editar y cómo editar los cambios pertinentes, además de ser redundante en el propio código.





**PATRONES**

**Builder Pattern:** es un patrón de diseño que permite la creación de elementos complejos dividiendo su proceso en una guía de paso a paso, lo que permite producir diferentes tipos de representación del objeto usando un proceso similar sin necesidad de pensar un método completamente nuevo para su elaboración. También, como principal función, es expandir los elementos complejos, permitiendo múltiples configuraciones mientras se separa la lógica de construcción del objeto de su representación final. A su vez, descomprime y optimiza los constructores y objetos para hacer el trabajo más dinámico y menos tedioso.

Los componentes que hacen parte del Builder Pattern:

**Director:** Coordina el proceso de construcción utilizando un objeto Builder. Oculta los detalles específicos de cómo se construye el objeto y simplemente dirige al Builder.

**Builder (Constructor):** Define una interfaz para construir partes del objeto de manera individual y para ensamblar las partes en el objeto final.

**ConcreteBuilder (Constructor Concreto):** Implementa la interfaz Builder para construir y ensamblar las partes específicas del objeto.

**Product (Producto):** Representa el objeto complejo que se está construyendo. La estructura y el contenido del producto varían según el ConcreteBuilder utilizado.

