❌ методы или функции принимают большое количество аргументов

❌ сложные логические проверки

❌ длинные строки кода, помещенные в одну строку

❌ синтаксически идентичный дублирующийся код (такой код может иметь разное форматирование)

❌ функции или методы, выполняющие много задач

❌ классы или компоненты, включающие много функций или методов

❌ сложная логика в одной функции или методе

❌ функции или методы с большим количеством инструкций return

❌ код с одинаковой структурой (названия переменных могут отличаться)

* Избегай сложного состояния за счет удаления лишнего.
* Передавай банан, а не гориллу, держащую банан, вместе с джунглями (старайся передавать примитивы в качестве пропов).
* Компоненты должны быть маленькими и простыми - принцип единственной ответственности!
* Дублирование лучше плохой абстракции (избегай преждевременных / неуместных обобщений).
* Решай проблему передачи пропов за счет композиции. Context не единственное решение проблемы распределения состояния.
* Разделяй большие useEffect на несколько небольших.
* Извлекай логику в хуки и вспомогательные функции.
* Большой компонент лучше разделить на logical и presentational компоненты (не обязательно, зависит от ситуации).
* Старайся, чтобы в зависимостях useEffect, useCallback и useMemo находились только примитивы.
* Зависимостей этих хуков не должно быть слишком много.
* Вместе нескольких useState лучше использовать один useReducer, особенно в случае, когда некоторые значения состояния зависят от других значений или предыдущего состояния.
* Context не должен быть глобальным для всего приложения. Он должен находиться максимально близко к компонентам, потребляющим контекст. Такая техника называется "размещением совместного состояния" (state collocation).