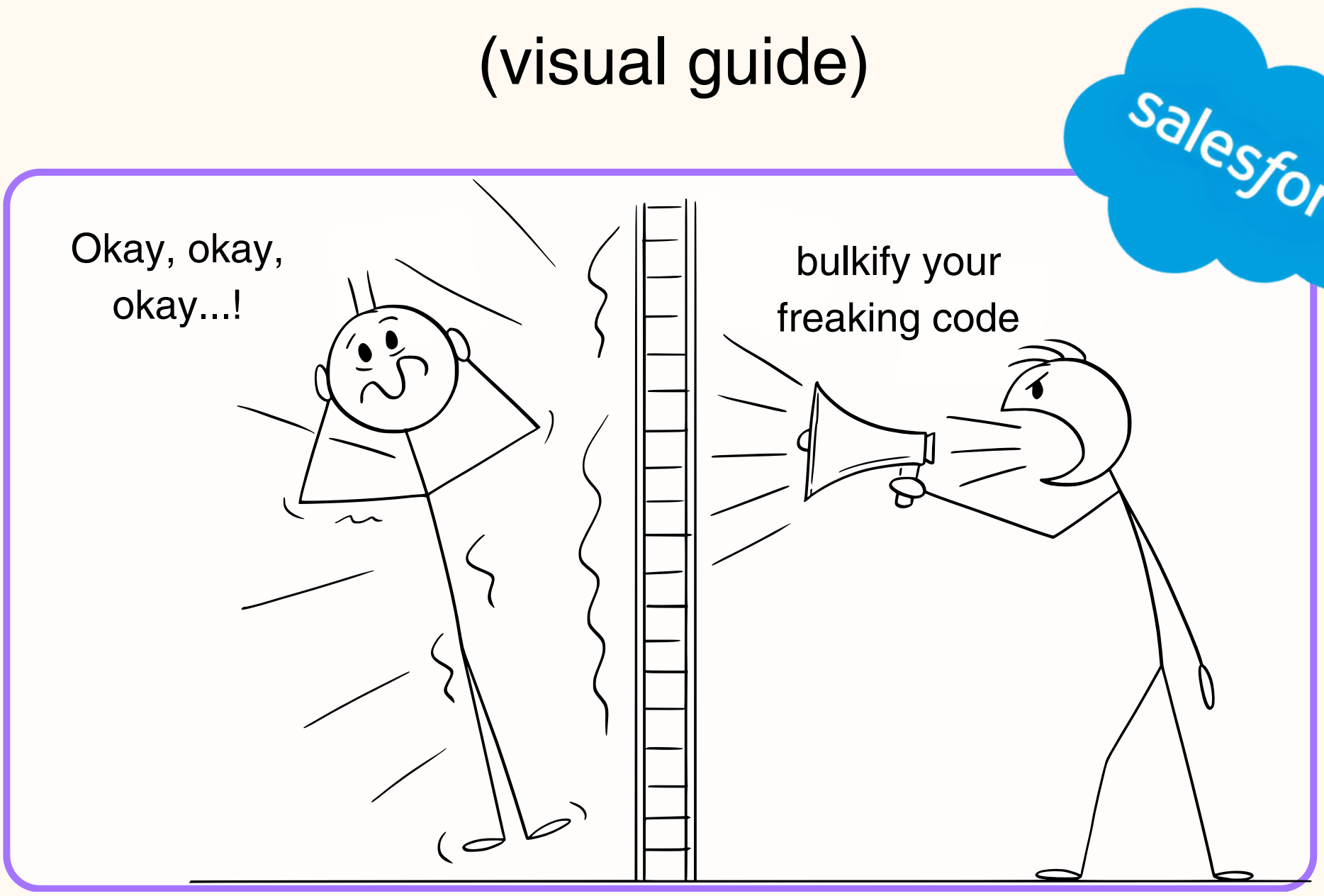
10 APEX

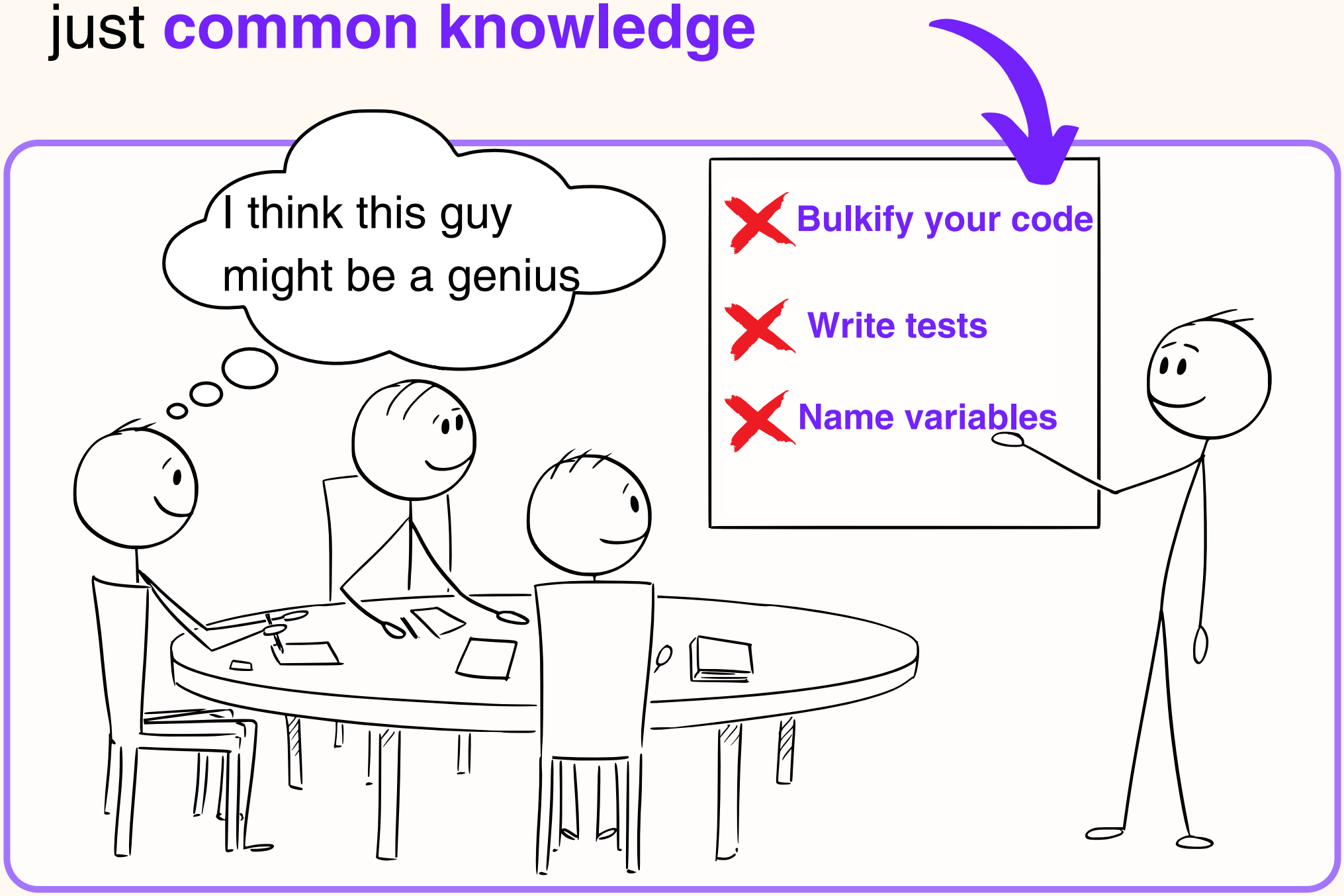
BEST-PRACTICES

(visual guide)



Igor Kudryk

Many **best-practices** that people share is just **common knowledge**



But what about **real** best-practices?

#1: Implement the **Switch-Off**

If you deploy Apex to **production** and it doesn’t go as planned...

... you need to be able to **switch off** your code

(especially for triggers)

Create a **custom metadata** type to store the **toggle** for the code

```
Switch__mdt setting = Switch__mdt.getInstance(
  'AccountTriggerHandler'
);
```

```
if (setting.Is_Active__c) {
  AccountTriggerHandler.process(accounts);
}
```

Run you code only if metadata is **enabled**

#2: Don’t write comments

I’ve seen **many times** something like this:

```
/**
* @description Sums up numbers
* @author Igor Kudryk
* @date 32 March 1995
* @return sum of numbers
*/
public Integer sumNumbers() {
...
}
```

Here’s why it’s very **bad**

What method does and returns you can read from the **method name**

```
/**
 * @description Sums up numbers
 * @author Igor Kudryk
 * @date 32 March 1995
 * @return sum of numbers
 */
public Integer sumNumbers() {
...
}
```

And **who** did the change and **when** should be handled by **Git**

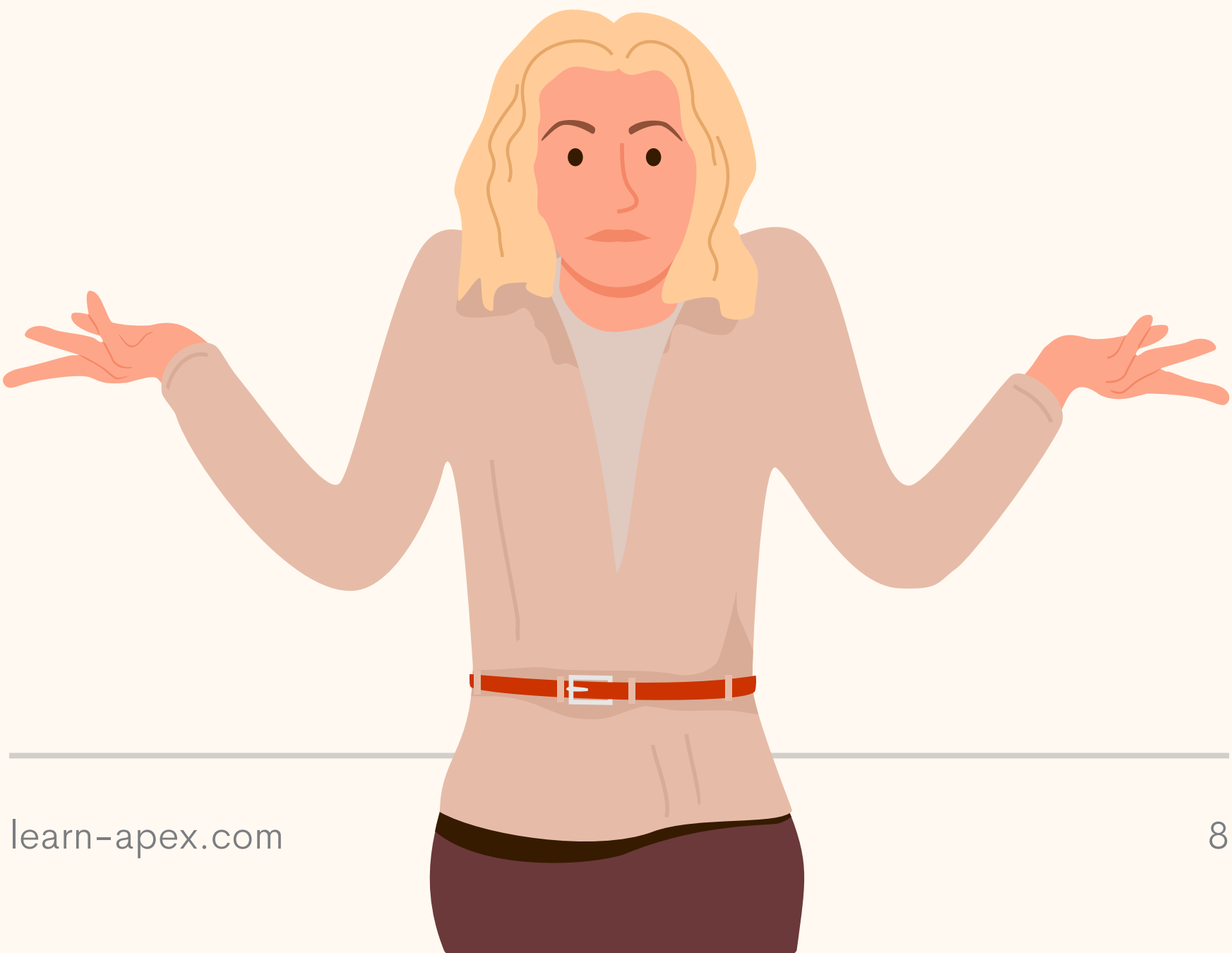
You should write comments only in **very few cases**:

#1 To explain **weird solutions**

#2 To warn about **workarounds**

#3 To give **context** for the method

A good mental model is to write comments that answer “**Why not?**” question.



#3: Don’t use **@Future**

Unless you really have to, use **Queueables** instead of **@Future** methods:

#1 They allow **job chaining**

#2 You can **pass objects**

#3 They have **finalizers**

... and much more.

Queueables are just way **better**.

#4: Don’t avoid **nested loops**

There is this misconception:

“*I shouldn’t use nested loops, because they are **quadratic** complexity*”

And this is **true**... BUT

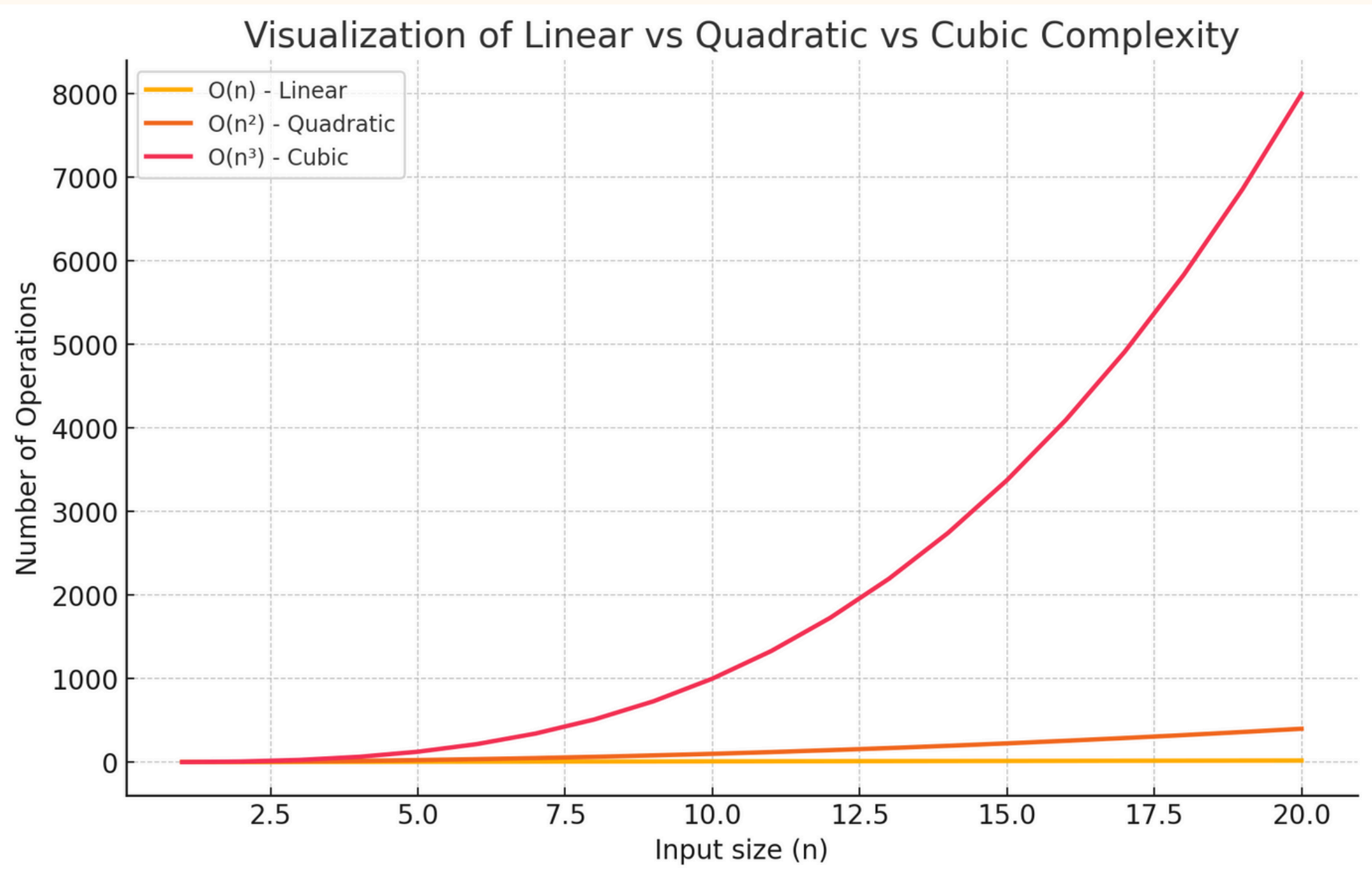
Loop Type	Time Complexity
Simple Nested Loop	$O(n^2)$
Nested with Different Bounds	$O(n \times m)$
Nested with Decreasing Inner Loop	$O(n^2)$
Nested with Constant Inner Loop	$O(n)$
Nested Loops with Multiplicative or Divisive Iteration	$O((\log n)^2)$

You still can use nested loops if you know you won’t have **many records**

e.g. if you have 100 accounts and each has 10 contacts

= 1000 iterations = no issue

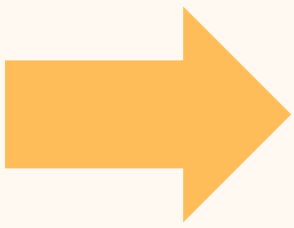
But that’s not true for **tripple nested loops**, they are absolutely **terrible** and have **cubic** time complexity



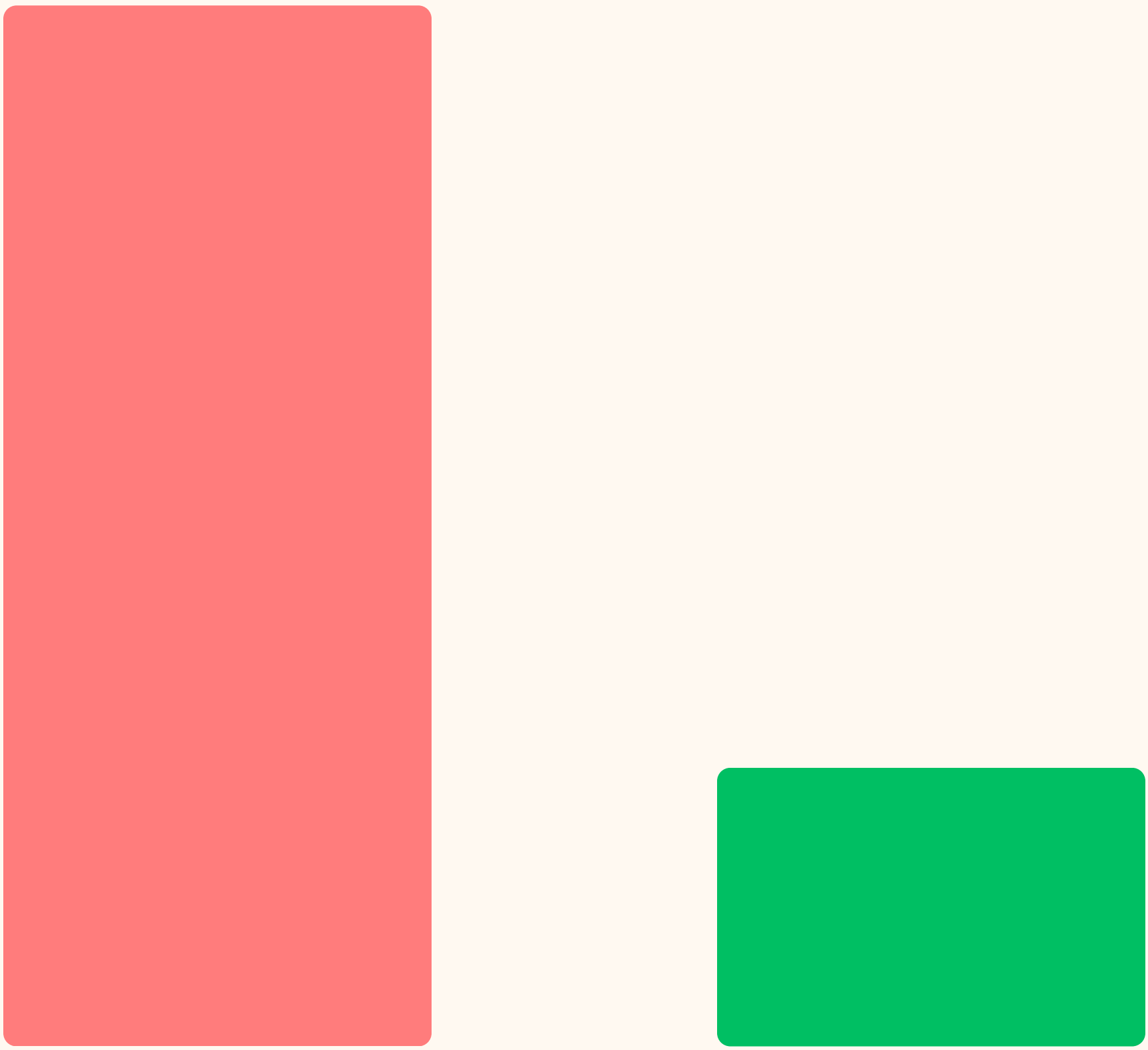
#5: Delete System.debug

System.debug doesn't contain any logic.

If it gets into production, you'll **slow down** your system **by a lot**.

Example 

CPU time



With debug

Without debug

#6: Don’t use “static”

People put **static keyword** everywhere.

But you should only do it, if you have **good reasons**.

Here’s only **3 cases**, where you need to use static

Use static keyword only in those cases:

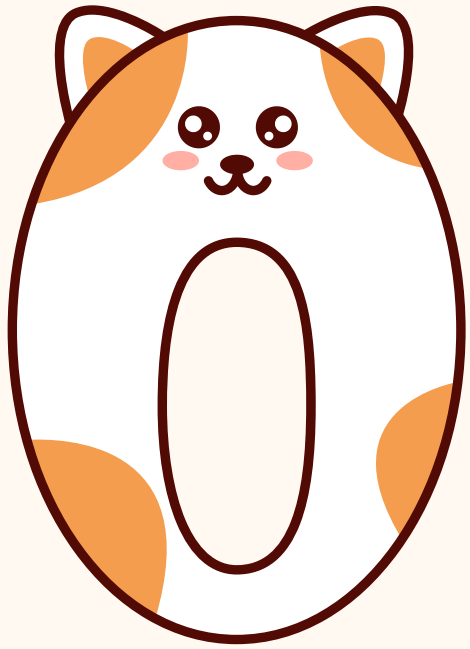
#1 Constant variables

#2 Method **logically makes sense** to be outside of the instance of the class

#3 Required by the **annotation**

#7: Return null

Returning null is totally fine

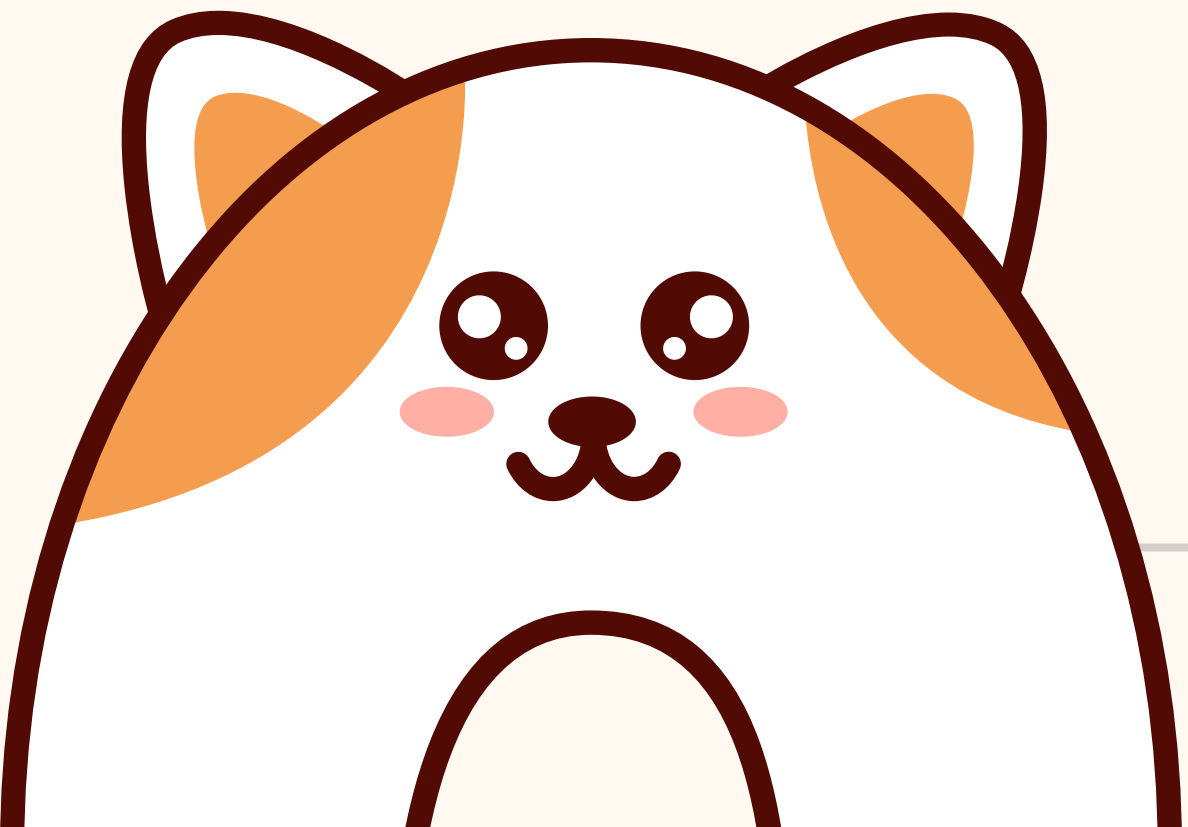


null = nothing

That’s just a way to say that **nothing** was the **result** of your **method execution**.

More nuanced:

- Returning null is **not bad or good**
- It’s just a way to **signal** that **nothing** resulted from your method
- But in **most cases you should return something** to prevent errors



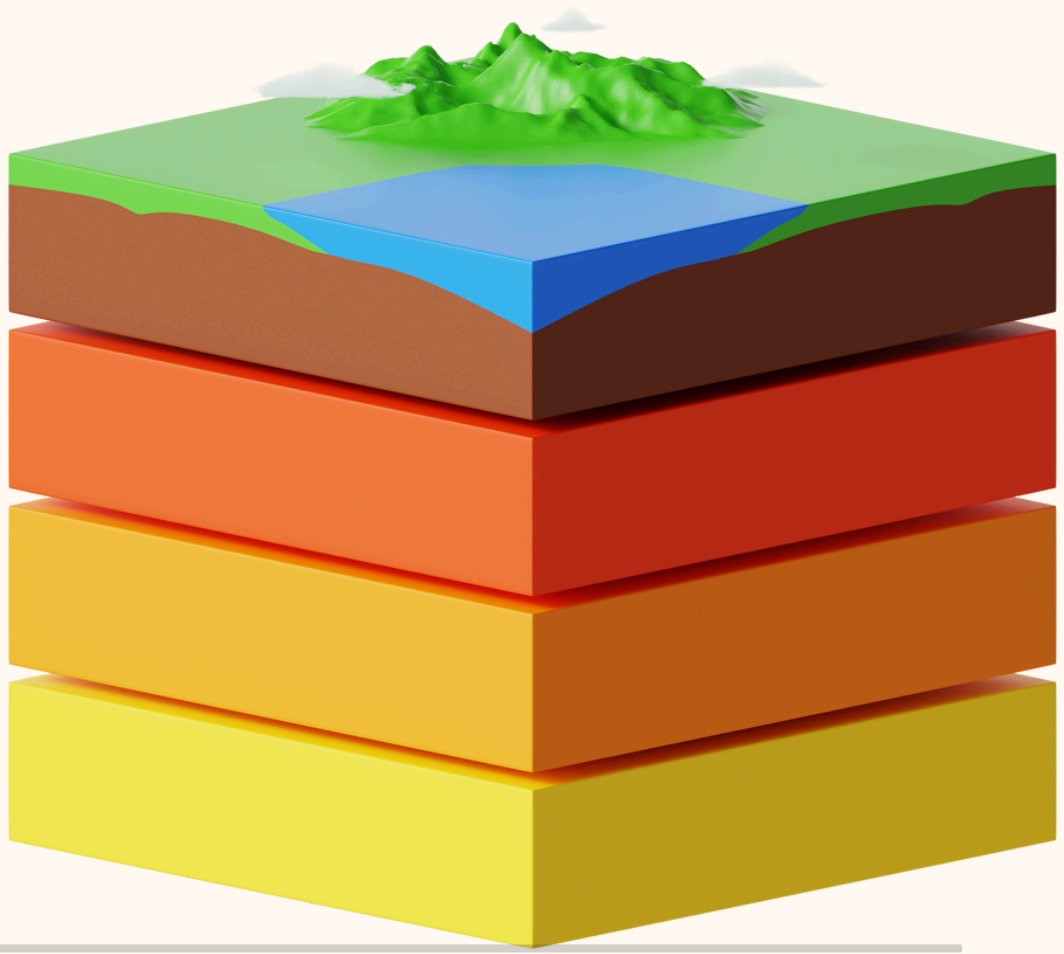
#8 Use Layers

in 80% of cases, you'd be good if you separate your code into a few layers:

#1 Orchestration layer

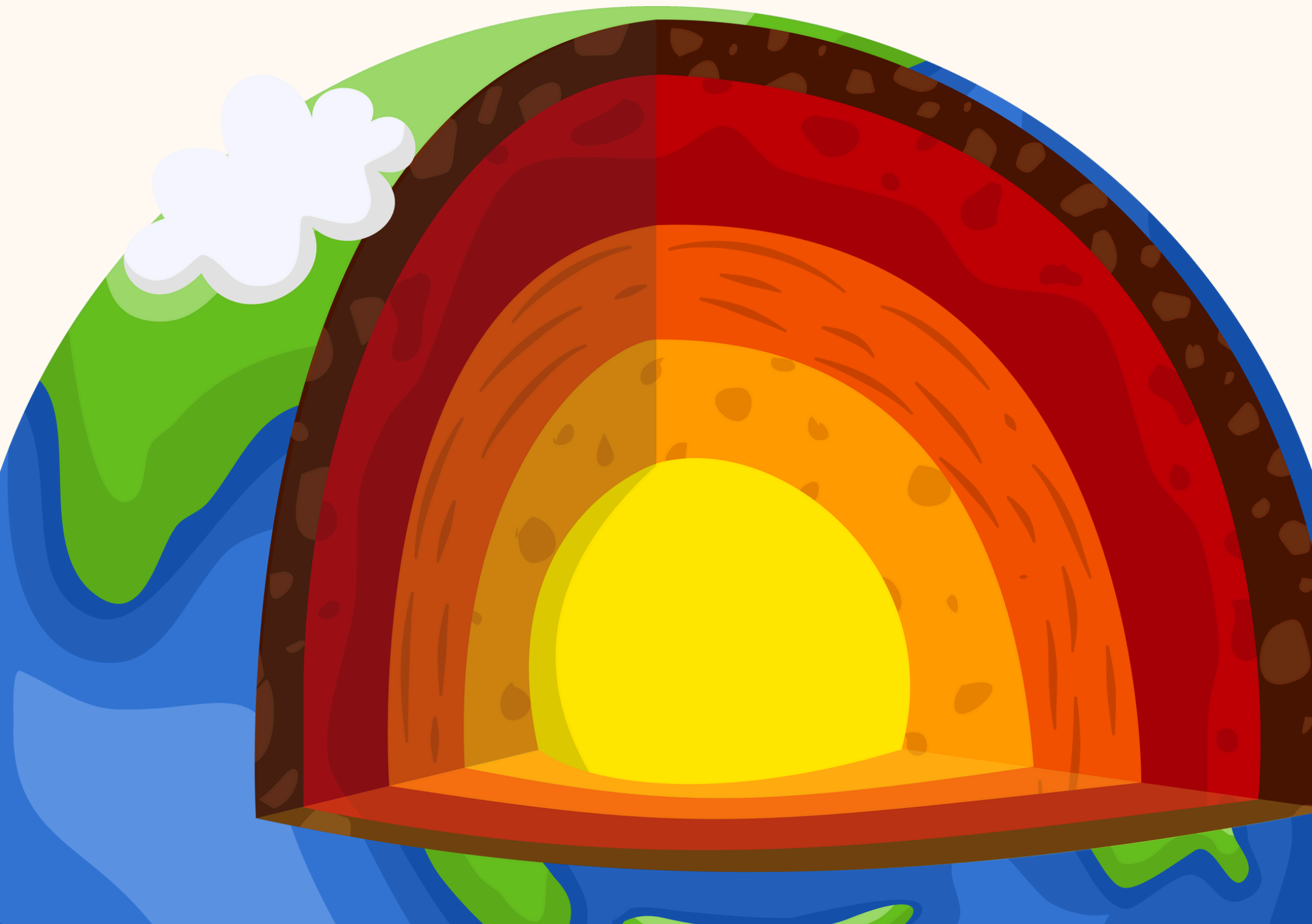
#2 Logic layer

#3 Selector layer



Using layers is just a fancy word for saying:

*“**Separate** your code in classes and **organise** them by function.”*

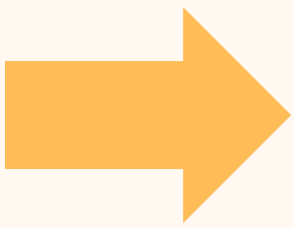


#9 Stop at “Good Enough”

Developers write code to **solve business problems.**

So you need to stop not when you solution is perfect.

But when it’s **good enough for the business**.

And that means 

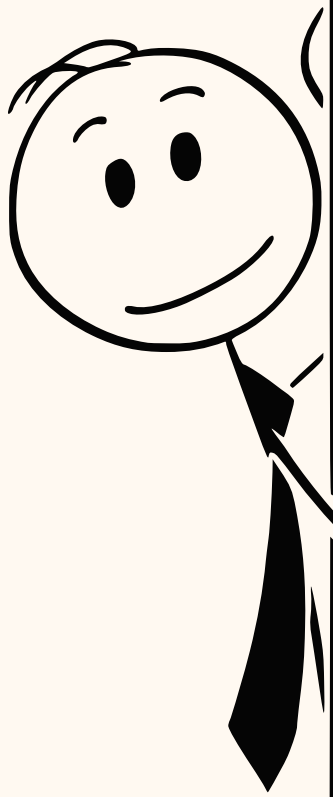
Focus on:

- ✓ Readability
- ✓ Clear, concise code
- ✓ Maintainability
- ✓ Scalability
- ✓ Targeted refactoring

Not important:

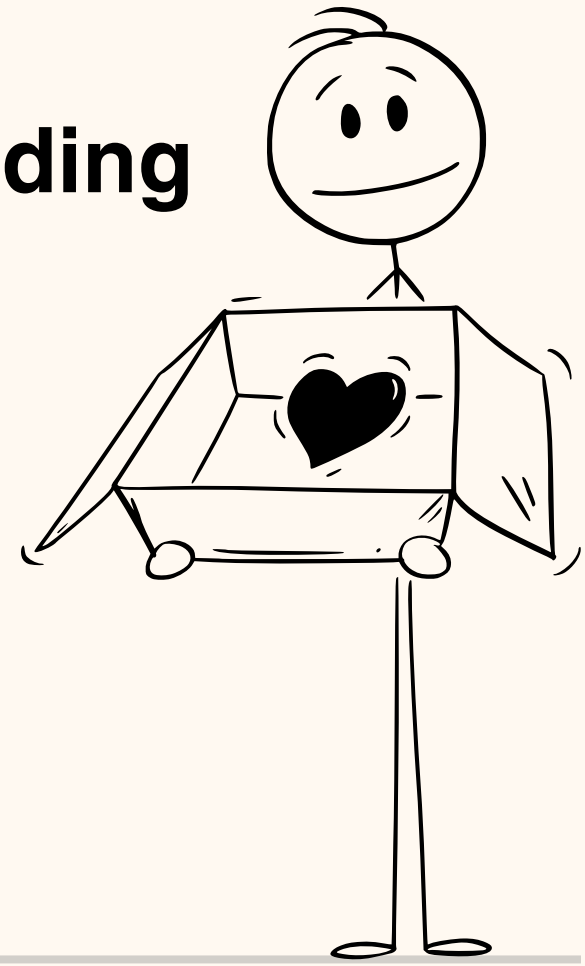
- ✗ Newest frameworks
- ✗ Endless refactoring
- ✗ Aesthetics
- ✗ Premature optimisation

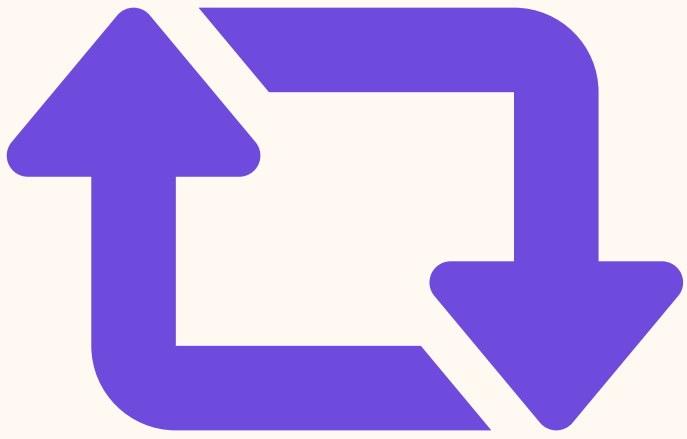
#10 Forget best-practices



They are **guidelines**,
not strict rules.

Thank you for reading





Reshare this post

It‘s the best thing you can do to help others understand best-practices in Apex

+ Follow

for more Apex stuff