MAKING BEAUTIFUL PYTHON CODE

Or Readability Counts

BY

JAIME BUELTA

Python community is quite aware of it, and choosing Python because code is readable is a very common. You’re doing it right just by showing here.

The sad truth

And problems happen... but until then

BOSS DOES NOT CARE

WORRIED ABOUT COSTS AND STUFF

CUSTOMERS DON'T CARE

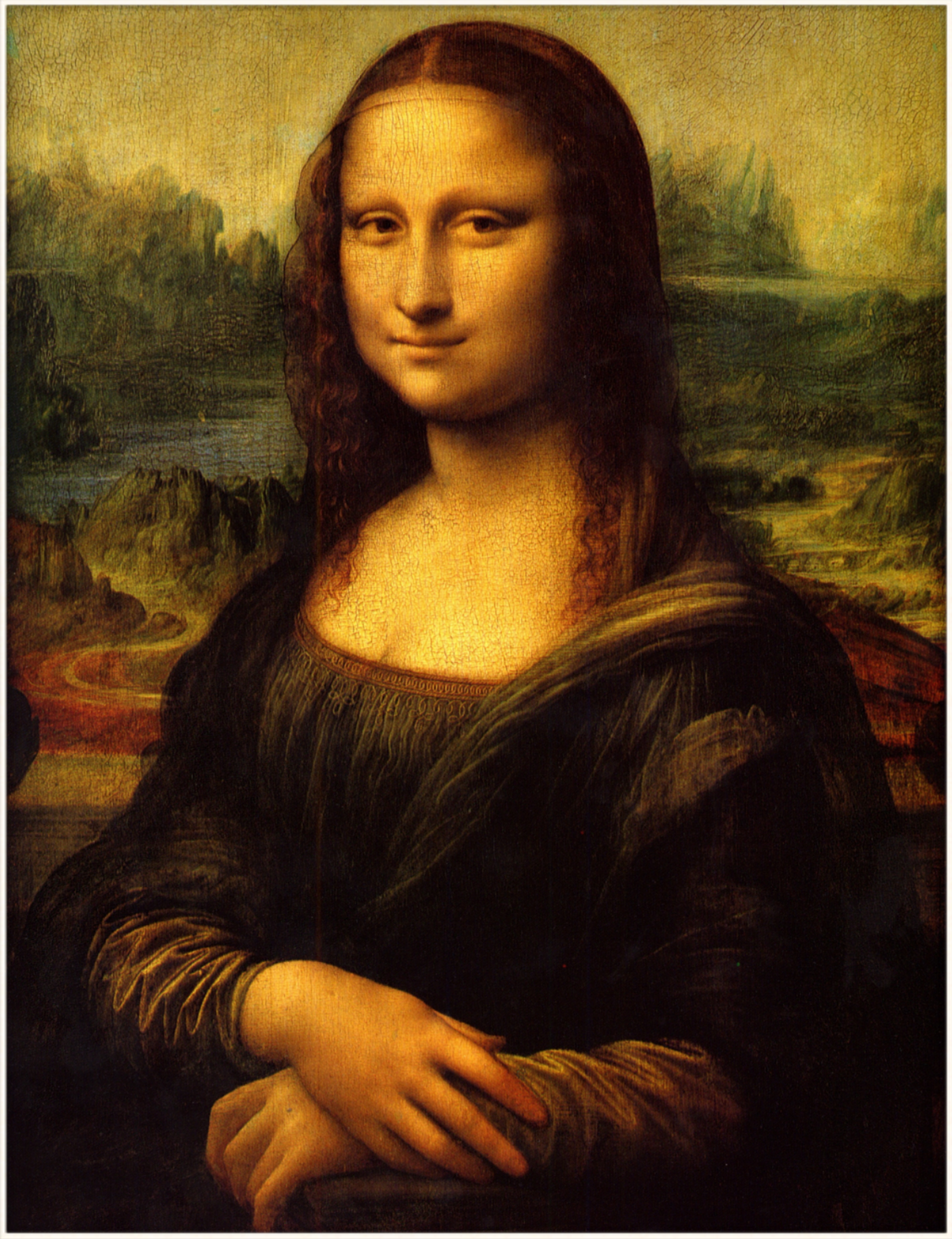
WORRIED ABOUT FUNCTIONALITY AND STUFF

It’s mostly an internal thing

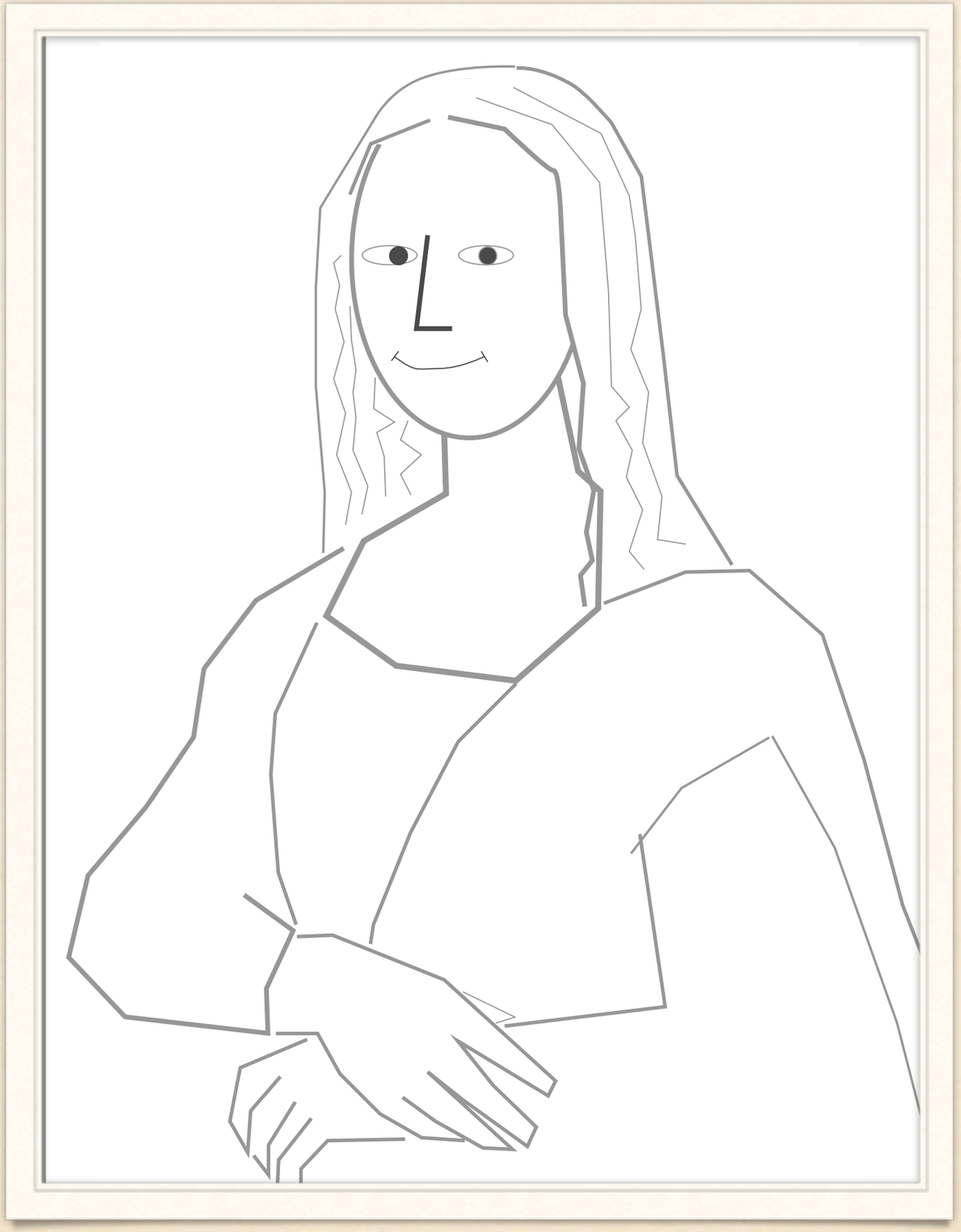
if you get used to it

MY DREAM

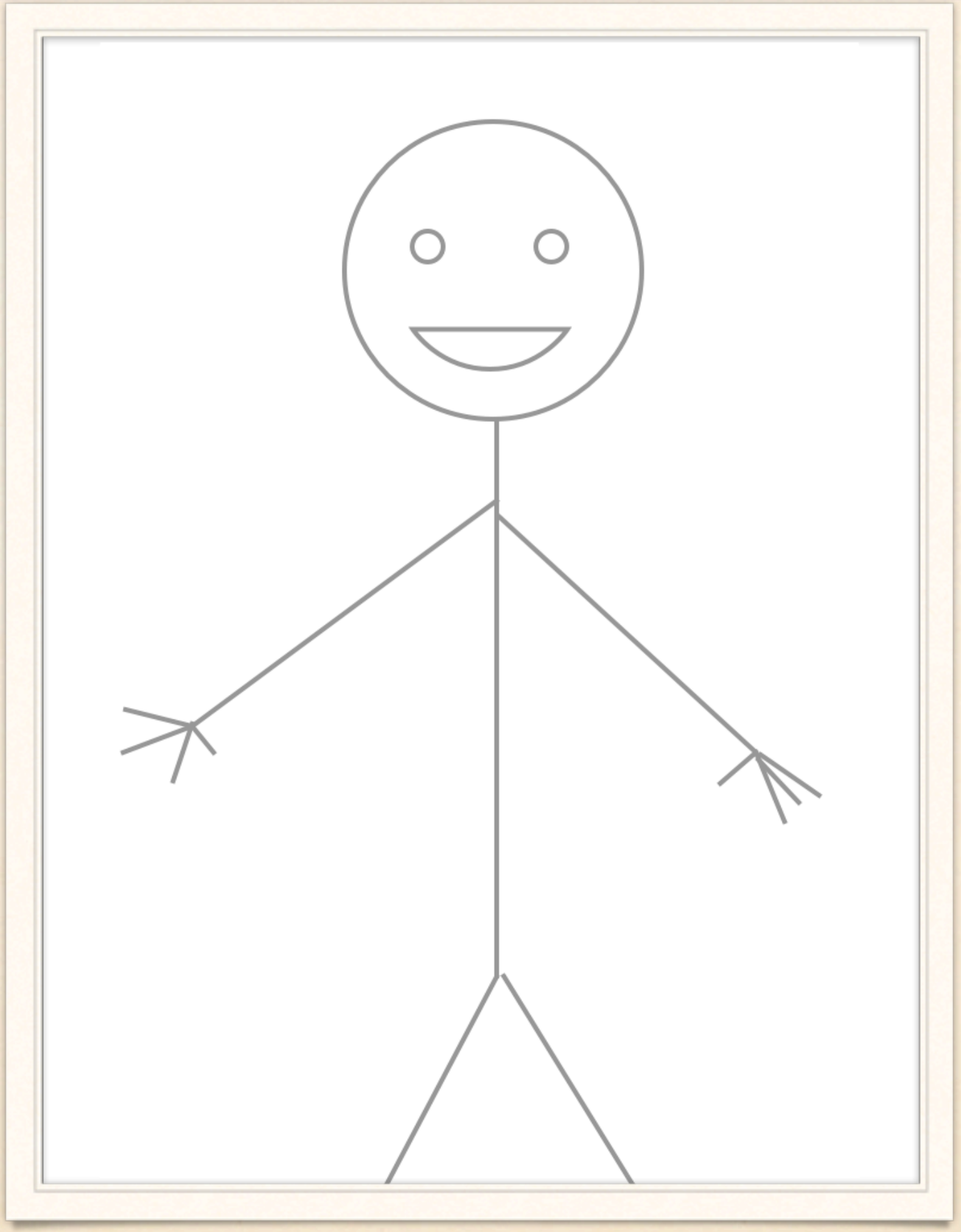
Stendhal Syndrome. Stendhal was a XIX century French author, who visited Florence, and was overwhelmed by all the beauty that surrounded him. I have a dream, that at some point I'll be able to create code so beautiful, that people will stare at it and feel anxiety.



I try to do this, and put a lot of effort into going there



But, at best this is my result. Not very impressive.



Most of the time is more like this

The alternative is this. Which also makes you stare and sweat, but for different reasons

Especially if the code changes as often

CODE IS ALWAYS CHANGING

CODE NEEDS TO WORK

That’s the most important consideration of all. Needs to do what’s supposed to and perform reasonably well. Sometimes, there will be some awful code. Again, no one external cares about the looks of the code.

And that's fine. Try to contain the ugly code and limit its scope. As the code changes, you can always come back and refactor. (The actor is Warren Brown)

WHAT MAKES BEAUTIFUL CODE?

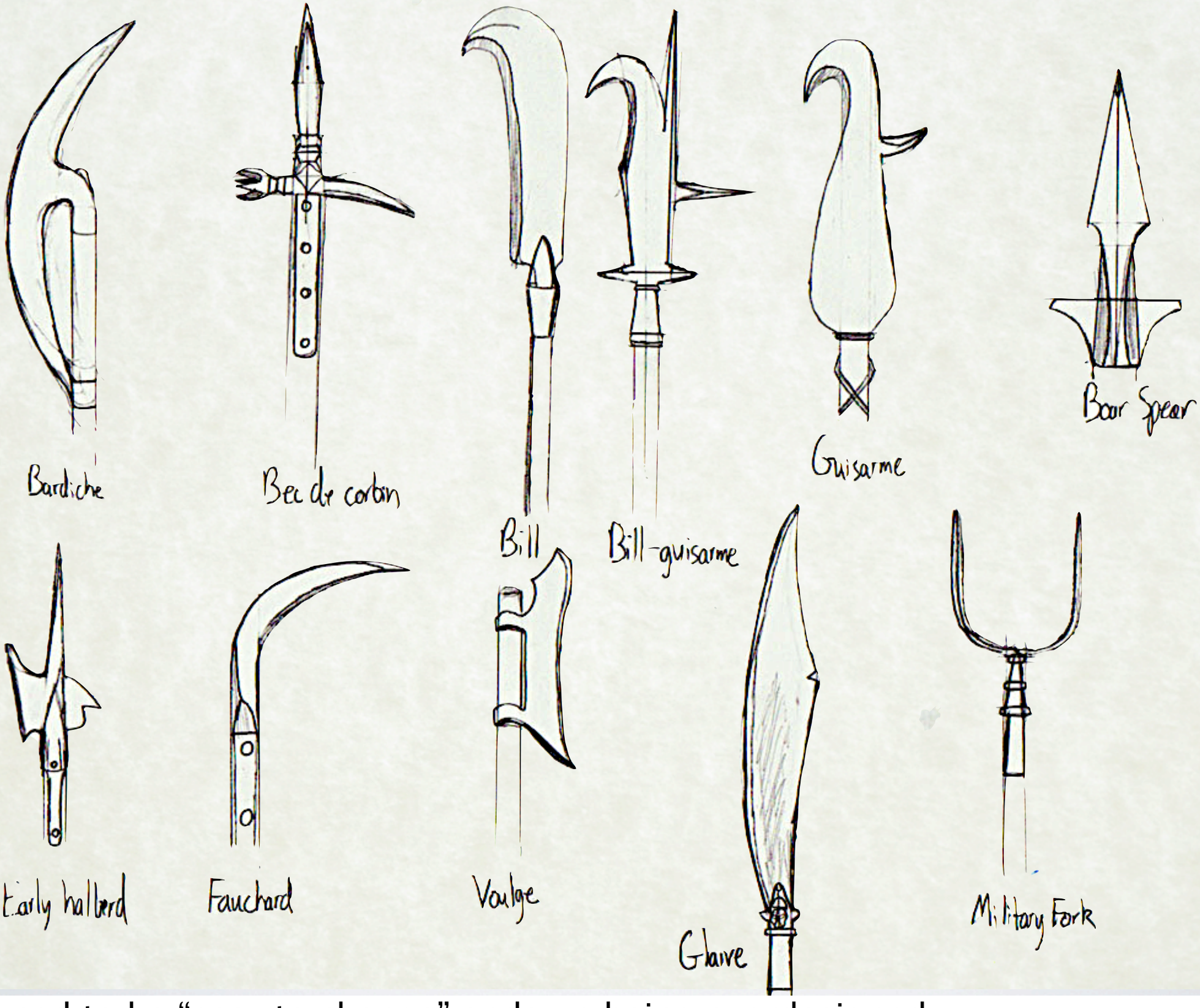
EASY TO UNDERSTAND

Remember that code is always changing

NOT THE OTHER WAY AROUND

Easy to change code is not always easy to understand (plugins, extensions, etc)

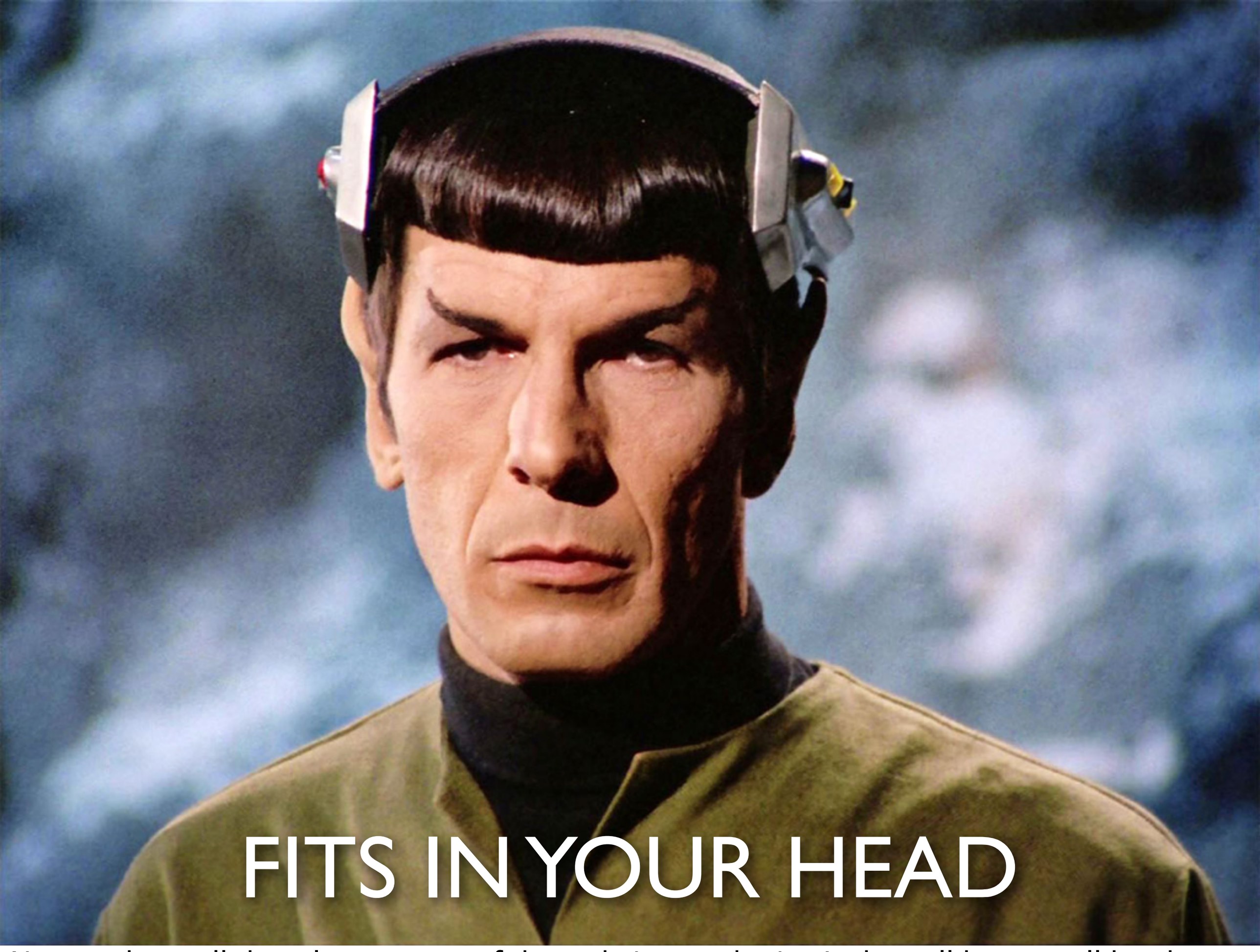
OVERDESIGN



Code designed to be “easy to change” ends up being overdesigned

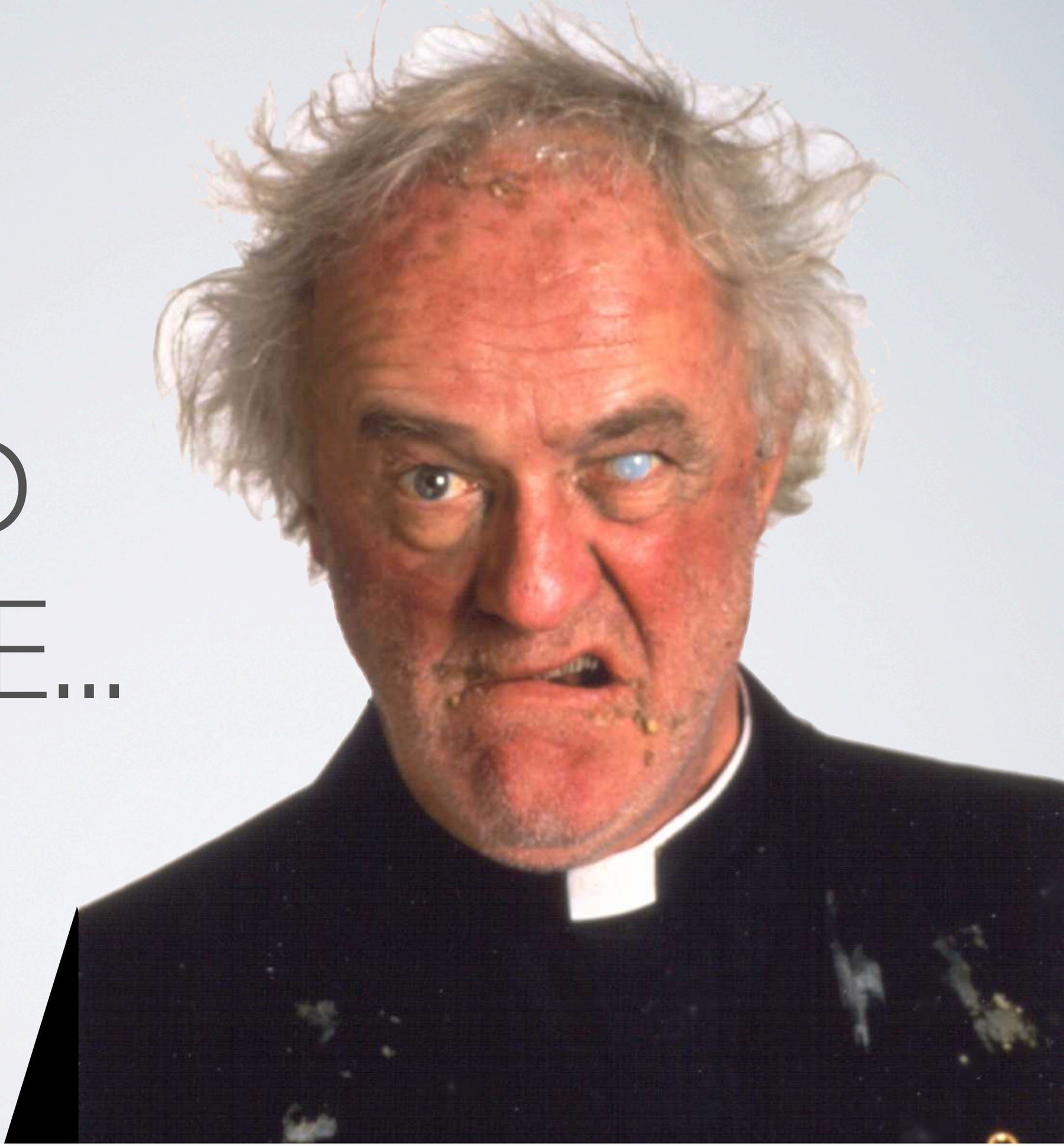
GOOD CODE

Everything is going fine. Aha, aha, aha. Actually, it is difficult to appreciate because it looks dull and simple and unimpressive most of the time.



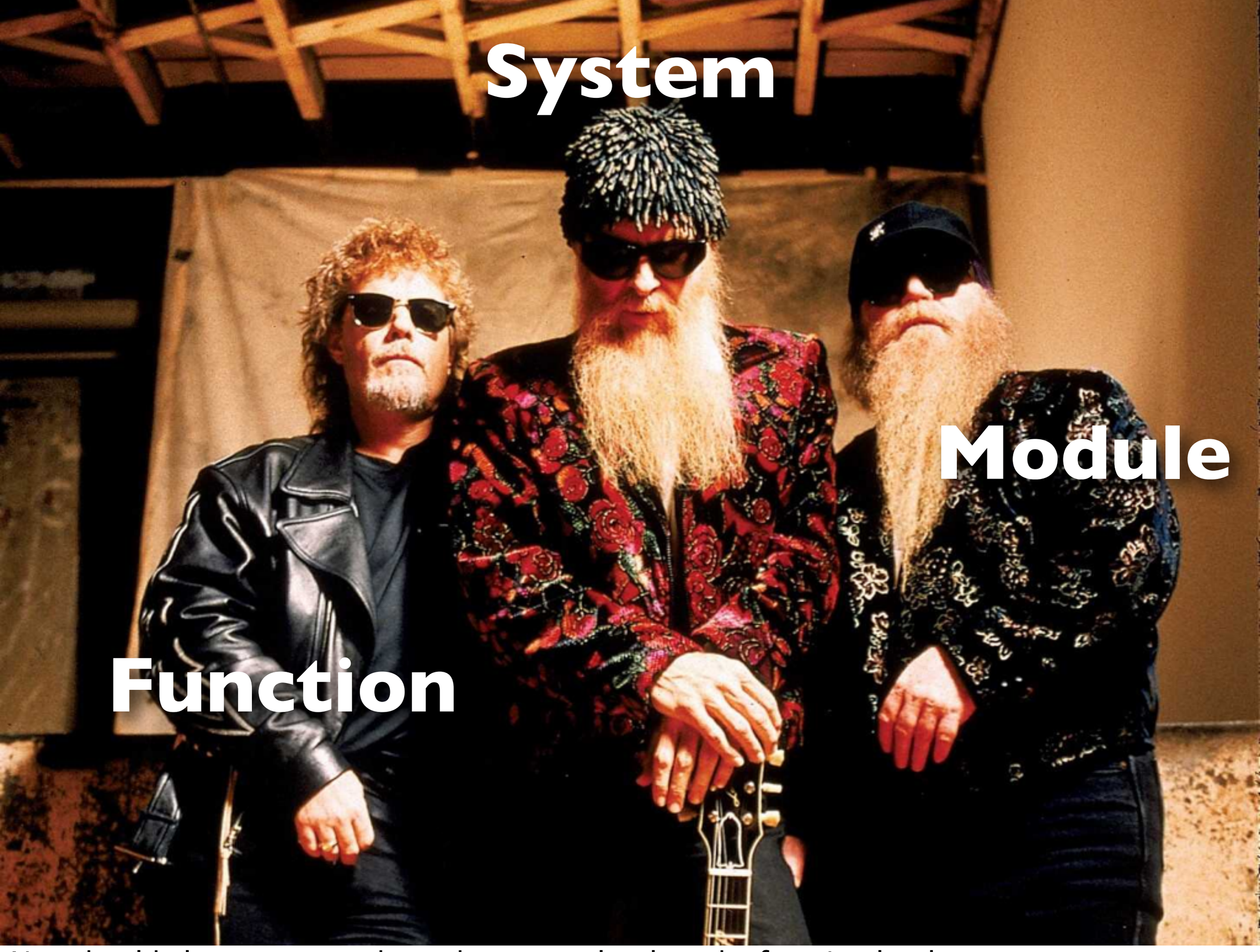
You can keep all the relevant parts of the code in your brain. And we all have small heads.

BAD CODE...



Will turn you into Father Jack

THREE LEVELS OF CODE



You should also comment about the system level on the function level

Disclaimer, Frank Beard is an awesome drummer

THREE PRINCIPLES FOR READABLE CODE

Rules of thumb. Those are guidelines, not rules.

LOCALITY

CONSISTENCY

VERBOSITY

LOCALITY

Keep related stuff together

CONSISTENCY

VERBOSITY

and non-related stuff separated. It helps keeping the relevant code for the task as small as possible.

LOCALITY

CONSISTENCY

Use patterns

VERBOSITY

Helps increasing the amount of code you can fit in your head

LOCALITY

CONSISTENCY

VERBOSITY

When in doubt, explain

Helps understanding the code

in other words...

Be as OBVIOUS as possible

No shit, Ackbar. Do you remember the Aha moment?

LOCALITY

Keep related stuff together (and not related stuff separated).

```
MSG_TEMPLATE = 'Template {date}: {msg}'
MORE_CONSTANTS...
....


def log_msg(message):
    formatted_msg = MSG_TEMPLATE.format(
        date=utcnow(),
        msg=this_message
)
print formatted_msg
```

```
def log_msg(message):
    MSG_TEMPLATE = '{date}: {msg}'
    formatted_msg = MSG_TEMPLATE.format(
        date=utcnow(),
        msg=message
    )
    print formatted_msg
```

But, what if I import that constant for a different module?

```
MSG_TEMPLATE = '{date}: {msg}'

def log_msg(message):
    formatted_msg = TEMPLATE.format(
        date=utcnow(),
        msg=message
    )
    print formatted_msg
```

Sometimes it will be unavoidable to move things around, as they'll be used by different modules/methods/functions

Other example: Convenience functions that are used only in one place

Related classes on the same module

Keep files short

Function used only in a module not in independent module

Avoid boilerplate

Separated business logic

Short meaning around 1K LOC. Bigger than that is usually a bad idea.

CONSISTENCY

1886

1890

1900

1905s

1940

1950

1969

1985

1987

1993

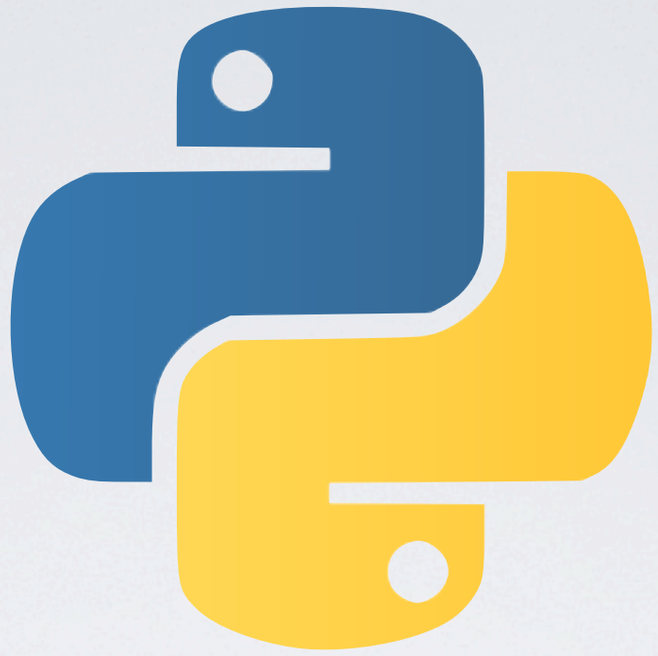
2003

2007

Use patterns. Those play very well with the way Python is designed, as you can use short and powerful patterns.

Humans are amazing recognizing patterns

(so good we tend to see too many)



PEP8 IS THE MOST IMPORTANT PATTERN COLLECTION OUT THERE

And it is quite followed. 80 chars per line, camelCase, CAPS for constants, etc. There are a few tools to help follow the rules (pyflakes)

Abstract common operations

No getters and setters

List comprehensions

Decorators and with statement

No private methods

minimum number of private methods and getters and setters. Decorators are great to avoid boilerplate code

I LOVE MY BRICK!

syndrome

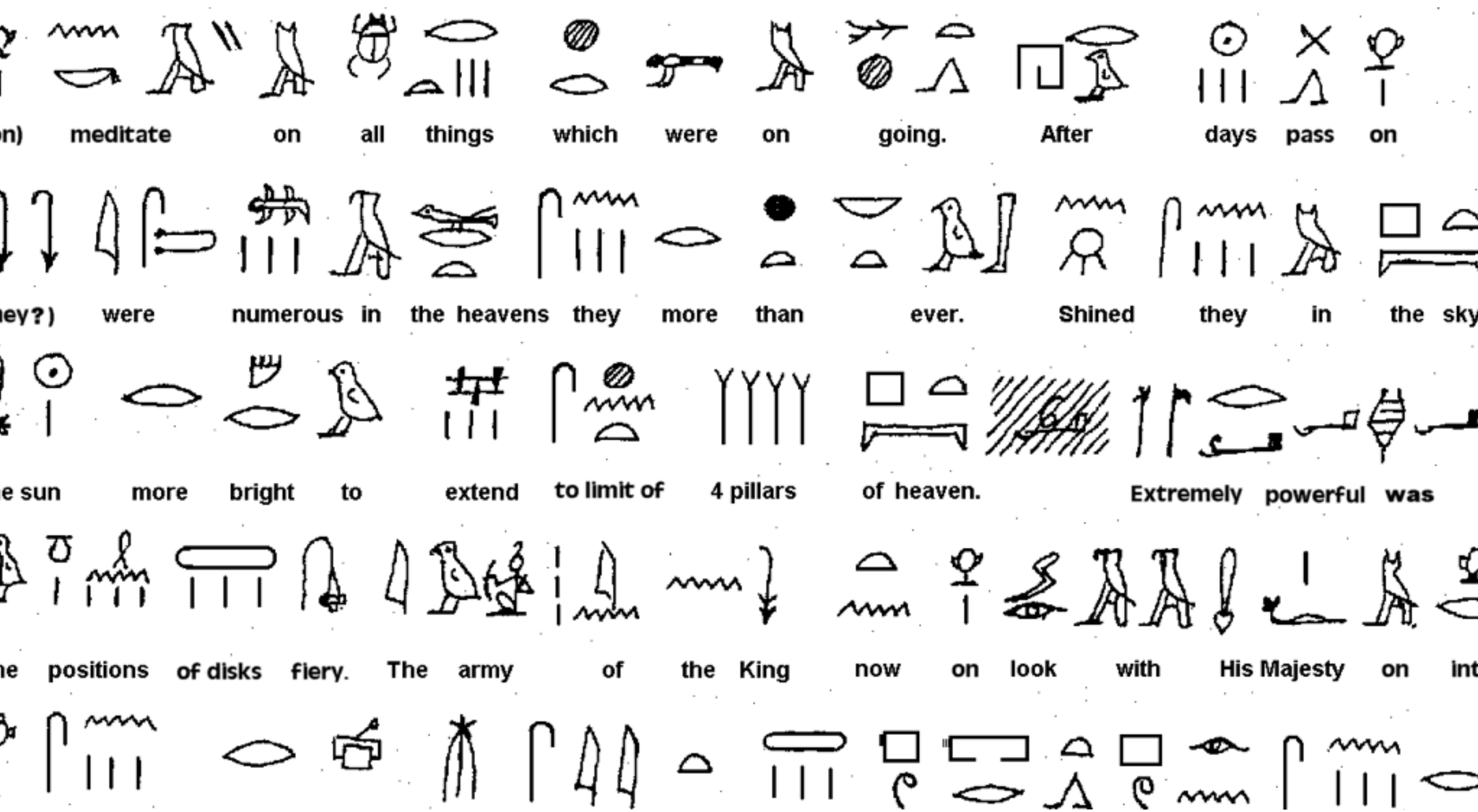
To be attached to a particular piece of code that you wrote. It's good to critically analyze what patterns do you use and if they can be better. Good example, getters and setters (in Java)

When in doubt, explain

```
data = "+RESP:GTTRI,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,"\
       "%s,%s,%s,%s,%s,%s,%s\0" % (imei, number,
                                   reserved_1, reserved_2,
                                   gps_fix, speed,
                                   heading, altitude, gps_accuracy,
                                   longitude, latitude, send_time,
                                   mcc, mnc, lac, cellid, ta,
                                   count_num, ver)
```

```
data = ('+RESP:GTTRI,{imei},{number},{reserved_1},'
        '{reserved_2},{gps_fix},{speed},{heading},'
        '{altitude},{gps_accuracy},{longitude},'
        '{latitude},{send_time}, {mmc}, {lac}, {cellid}'
        '{ta},{count_num}, {version}').format(
            imei=imei,
            number=number,
            reserved_1=reserved_1,
            reserved_2=reserved_2,
            gps_fix=gps_fix,
            speed=speed,
            heading=heading,
            altitude=altitude,
            gps_accuracy=gps_accuracy,
            longitude=longitude,
            latitude=latitude,
            send_time=send_time,
            mcc=mcc,
            mnc=mnc,
            lac=lac,
            cellid=cellid,
            ta=ta,
            count_num=count_num,
            version=ver,
    )
```

WHEN IN DOUBT, COMMENT



We can argue endlessly about what is the proper level of comment. But is always better one more comment than one less. You can always remove comments later.
And comments SHOULD be kept up-to-date. Their are part of the code. No excuses.

PUTTING ALL TOGETHER

```
results = []
for row in query_results:
    tag, value, updated = row
    if value and updated > last_time:
        TEMP = 'tag: {0}, value: {1}
        result = TEMP.format(tag,
                             value)
        results.append(result)
```

```
def format_result(row):
    tag, value, _ = row
    TEMP = 'tag: {0}, value: {1}'
    return TEMP.format(tag, value)

def interesting(row):
    _, value, updated = row
    return value and updated > last_time

results = [format_result(row)
           for row in query_results
           if interesting(row)]
```

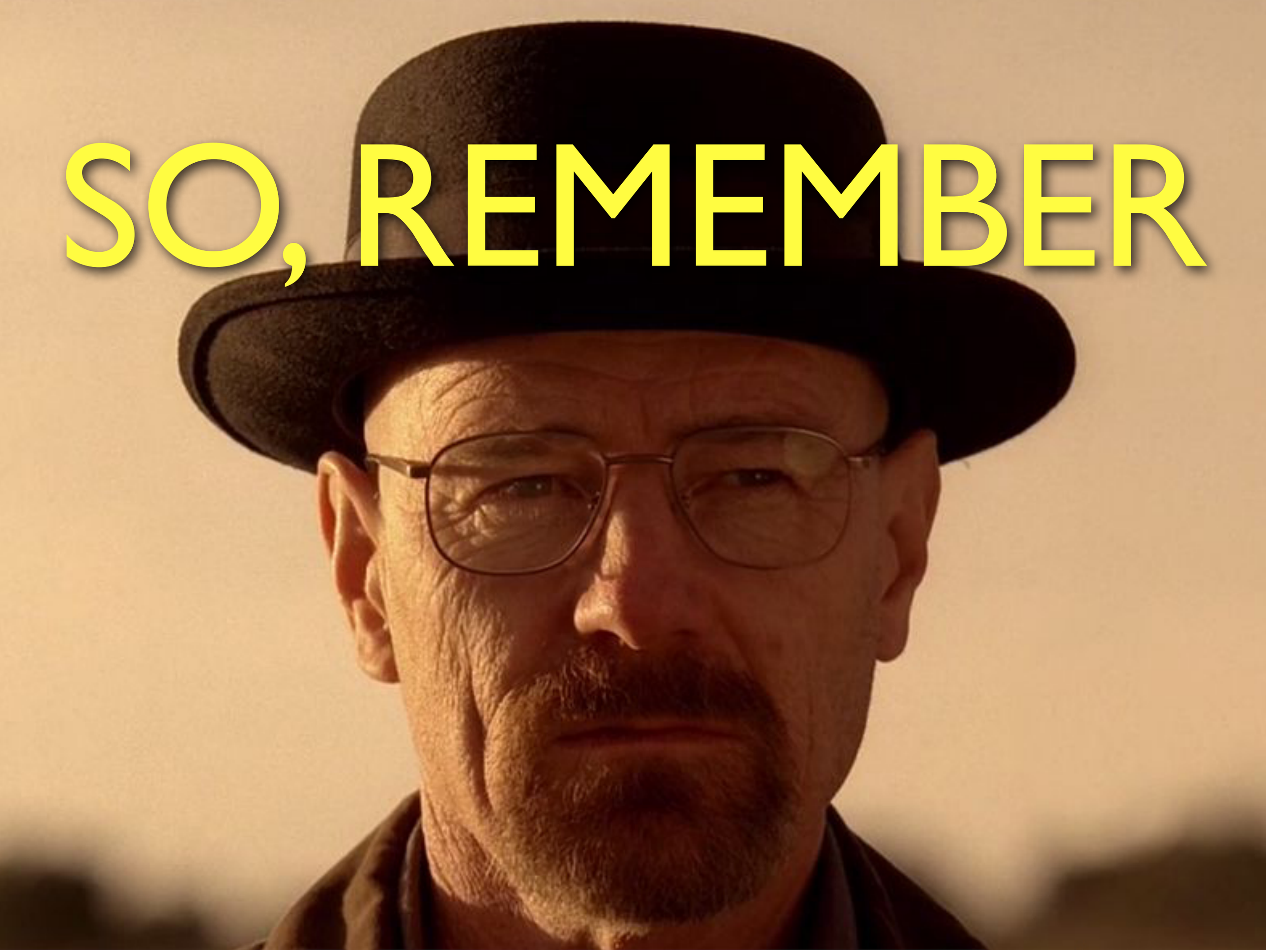
Locality (defs inside the function)
Consistency (use list comprehension)
Verbosity is not always comment, but chose to add description through names,etc

Being smart, specially on the function / line level has to obtain something (performance) and be commented. The default should be NO.

like winning a Golden Globe or something

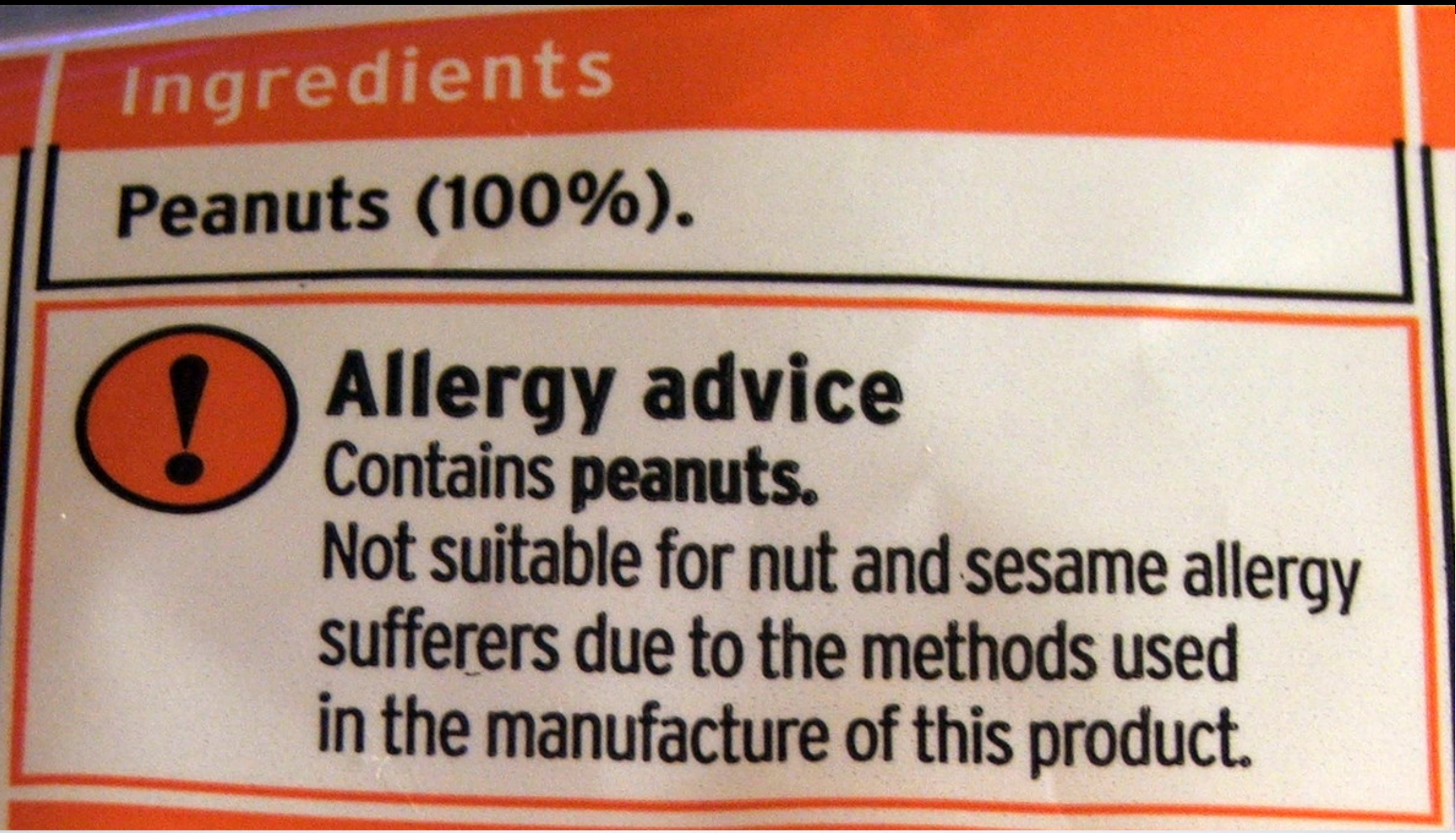

ONE DAY, SOMEONE WILL BE
SURPRISED WITH YOUR CODE

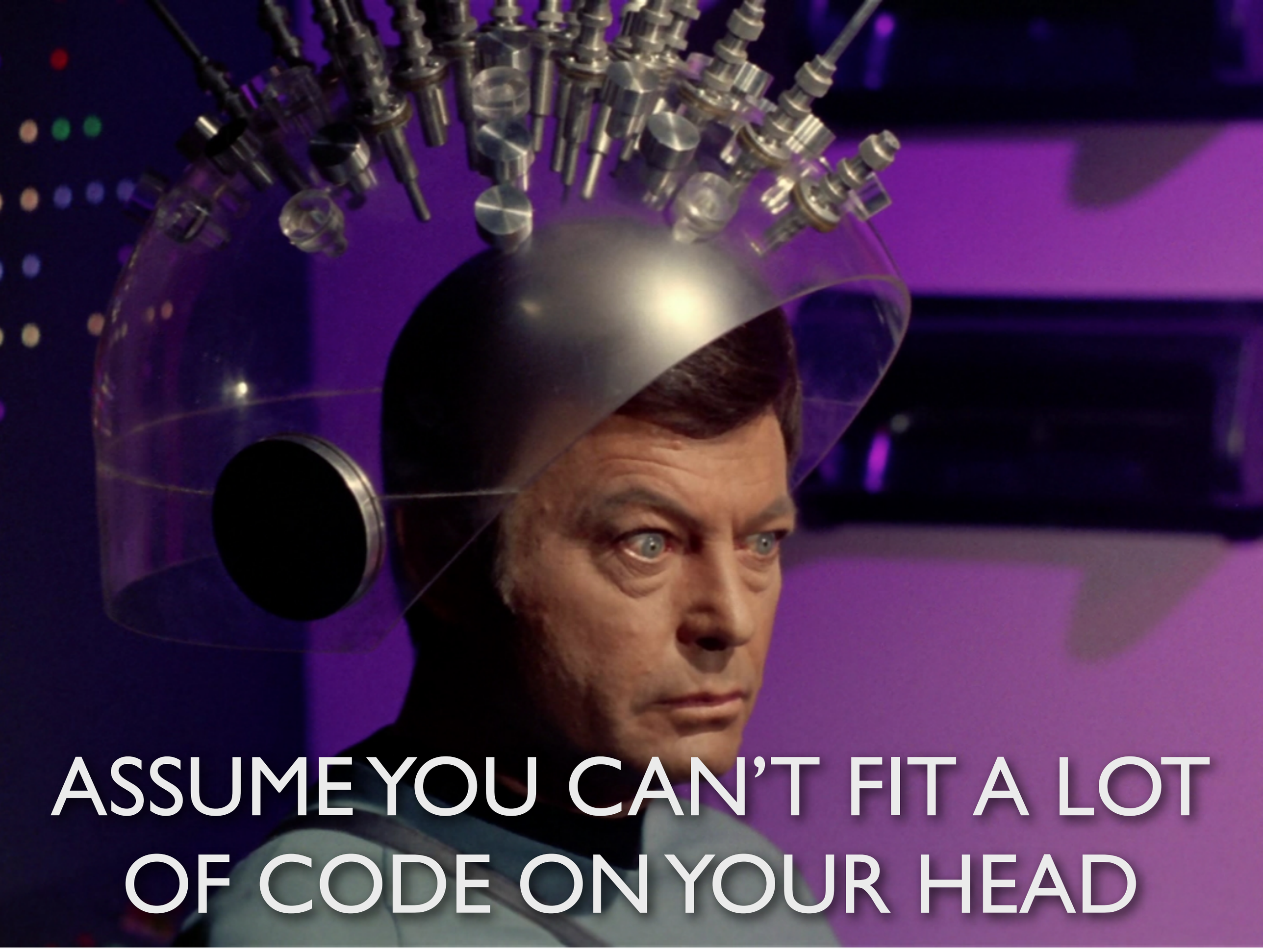

YOURSELF!!!


SO, REMEMBER

As much as possible. Try to actively think about it

BE OBVIOUS


Ingredients
Peanuts (100%).
Allergy advice
Contains peanuts.
Not suitable for nut and sesame allergy sufferers due to the methods used in the manufacture of this product.


ASSUME YOU CAN'T FIT A LOT
OF CODE ON YOUR HEAD

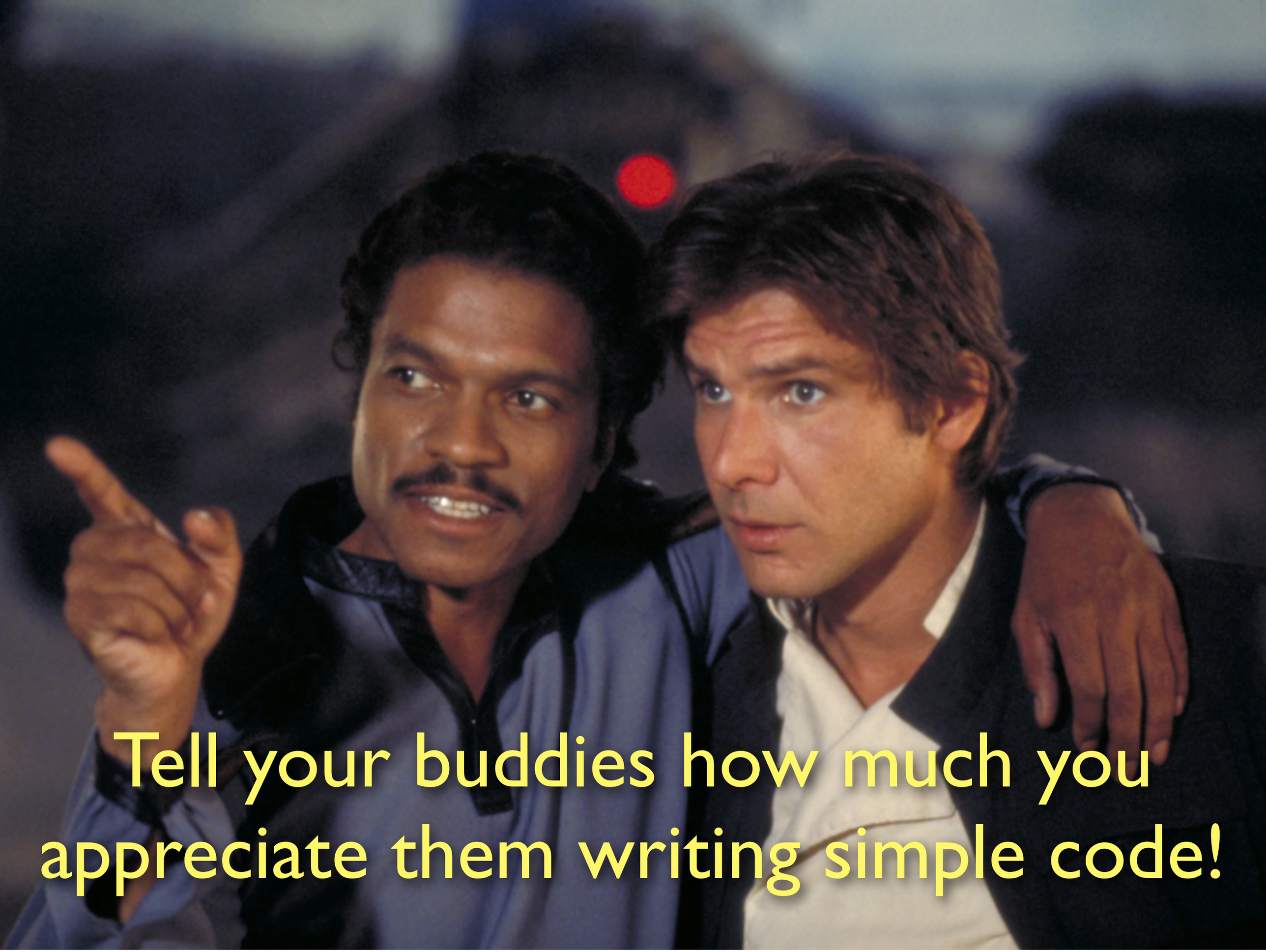
BE SMART IN SYSTEMS, NOT IN FUNCTIONS

LOCALITY

CONSISTENCY

VERBOSITY

Locality: Keep stuff related together and not-related stuff not together
Consistency: User patterns
Verbosity: When in doubt, explain


Tell your buddies how much you
appreciate them writing simple code!

Thanks for your attention !

Any Questions?

@jaimebuelta 

www.wrongsideofmemphis.com