Clean code

Petr Pošík

Department of Cybernetics, FEE CTU in Prague

EECS, BE5B33PRG: Programming Essentials, 2015

Based on:

- PEP 8 (https://www.python.org/dev/peps/pep-0008/).
- Robert C. Martin: Clean Code (http://www.amazon.com/Clean-Code-Handbook-Software-Craftsmanship/dp/0132350882)

Why?

- Code is read much more often than it is written.
- Readability counts.

What is "clean code"?

Bjarne Stroustrup

- author of C++ language, and “The C++ Programming Language” book

> *I like my code to be **elegant and efficient**. The logic should be **straightforward** to make it hard for bugs to hide, the **dependencies minimal** to ease maintenance, error handling complete according to an articulated strategy, and **performance close to optimal** so as not to tempt people to make the code messy with unprincipled optimizations. **Clean code does one thing well**.*

Grady Booch

- author of the book “Object Oriented Analysis and Design with Applications”

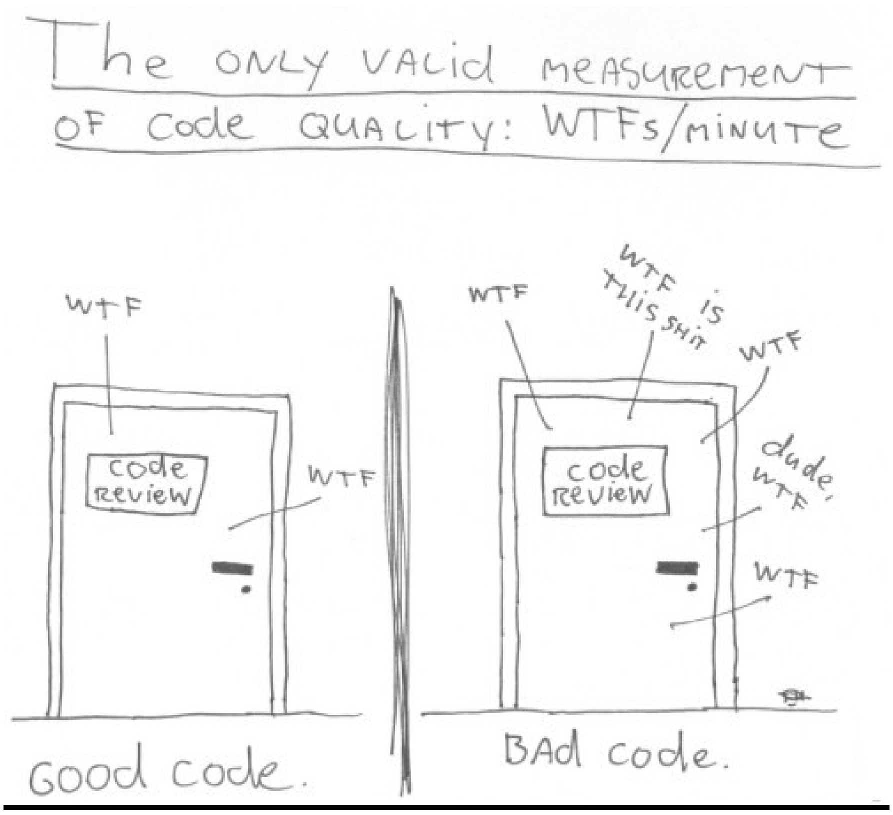
> *Clean code is **simple and direct**. Clean code **reads like well-written prose**. Clean code **never obscures the designer’s intent** but rather is full of **crisp abstractions** and **straightforward lines of control**.*

Dave Thomas

- founder of OTI (part of IBM from 1996), Eclipse godfather.

> *Clean code **can be read, and enhanced by a developer other than its original author.** It has **unit and acceptance tests**. It has **meaningful names**. It provides one way rather than many ways for doing one thing. It has **minimal dependencies**, which are explicitly defined, and provides a **clear and minimal API**.*

The only valid measurement of code quality



Python Enhancement Proposal (PEP) 8

- PEP 8 (https://www.python.org/dev/peps/pep-0008/) is a general **Python style guide**.
    - Consistency with PEP 8 is important.
    - Consistency within a project is more important.
    - Consistency within one module or function is most important.
    - Know when to be consistent - sometimes the style guide just doesn't apply.

Reasons to ignore a particular guideline

- When the guideline makes the code actually less readable.
- For consistency with the surrounding code that also breaks the guideline (maybe for historic reasons). But maybe this is an opportunity to clean the messy code.
- Because the code is older than the guideline and there is no other reason to modify the code.
- When the code needs to remain compatible with the older versions of Python that does not support the feature recommended by the style guide.

Code layout and organization

- Use 4 spaces (not tabs) for indentation.
- Limit the line length to 79 characters, 72 for docstrings and comments.
- Default encoding is UTF-8; otherwise specify encoding e.g. as
  - `# -*- coding: latin_1 -*-`
  - see PEP 263 (https://www.python.org/dev/peps/pep-0263/) for details
- Place imports at the top of the file, each module on separate line.
- Keep function definitions together.
- Separate top-level function and class definitions by 2 blank lines; separate nested functions and method definitions inside a class by 1 empty line.
- Keep top-level statements, including function calls, together at the bottom of the program.

Comments and docstrings

- Write your comments in English, unless you are 120 % sure that the code will never be read by people who do not speak your language.
- Use **docstrings** (see PEP 257 (https://www.python.org/dev/peps/pep-0257/)) to document public modules, functions, classes, and methods.

Comments

- Clean code needs no comments. Almost.
- Comments compensate for our **failure to express ourselves** in the programming language. Compare:

```
# Check if the employee is eligible for full benefits
if (employee.flags & HOURLY_FLAG) and (employee.age > 65):
  ...
```

versus

```
if employee.is_eligible_for_full_benefits():
  ...
```

Comments (cont.)

- Comments that contradict the code are worse than no comments!
- **Good comments**:
  - explanation, clarification; e.g.
    - `width += 1   # Compensate for frame border`
  - emphasis, warnings
  - TODOs
- **Bad comments**:
  - old, invalid, redundant, misleading comments
  - code commented out
  - non-local or irrelevant information

Names

- Use
  - `lowercase_with_underscores` for variables, functions, modules and packages,
  - `CamelCase` for classes and exceptions, and
  - `CAPITAL_LETTERS_WITH_UNDERSCORES` for "constants".
- Names of classes: **nouns (with adjectives)**:
  - `Customer, WikiPage, AddressParser, Filter, PrimesGenerator, ...`
- Names of functions/methods: **verbs (with objects)**:
  - `open, save, print, post_payment, delete_page, get_email, compute_salary, ...`

Names (cont.)

- A good name is **meaningful** and **reveals author's intent**. Compare:

```
d = 0  # Elapsed time in days
elapsed_time_in_days = 0
```

- To come up with a good name is not easy! Change the name, if you come up with a better one. Do not be afraid of long names!
- Use **named "constants"** instead of magic numbers in the code!

Functions and methods

- A function shall do one thing well.
- Functions shall be short (and even shorter), ca 5 lines:
  - They can hardly do more than 1 thing.
  - They can have meaningful and revealing name.
  - They can hardly contain nested `if` or `for` commands.
  - Code blocks inside `if`, `for`, ... must be short, ideally a single line.
- Short functions allow for testing individual parts of the algorithm.
- Function/method arguments:
  - Keep their number small (0, 1, 2, exceptionally 3).
  - Create a name that evokes the order of arguments.

Summary

- Whether your code is clean is subjective. You shall think about the code, about its meaning.
- Try to make it as readable and intention-revealing as possible.
- Well-chosen names make up for 80 % of clean code.
- Good names can be chosen when functions/methods are sufficiently short.
- Be DRY! Don't repeat yourself!

Notebook config

Some setup follows. Ignore it.

In [1]:
```
from notebook.services.config import ConfigManager
cm = ConfigManager()
cm.update('livereveal', {
              'theme': 'Simple',
              'transition': 'slide',
              'start_slideshow_at': 'selected',
              'width': 1268,
              'height': 768,
              'minScale': 1.0
})
```

Out[1]:
```
{'height': 768,
 'minScale': 1.0,
 'start_slideshow_at': 'selected',
 'theme': 'Simple',
 'transition': 'slide',
 'width': 1268}
```

In [2]:
```
%%HTML
<style>
.reveal #notebook-container { width: 90% !important; }
.CodeMirror { max-width: 100% !important; }
pre, code, .CodeMirror-code, .reveal pre, .reveal code {
    font-family: "Consolas", "Source Code Pro", "Courier New", Courier, monospace;
}
pre, code, .CodeMirror-code {
    font-size: inherit !important;
}
.reveal .code_cell {
    font-size: 130% !important;
    line-height: 130% !important;
}
</style>
```