```
(sdk) C:\Users\asanka_wasala>python
Python 3.7.10 (default, Feb 26 2021, 13:06:18) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
>>> quit()
```

Clean Code Strategies

for Team Leads

Asanka Wasala, PhD

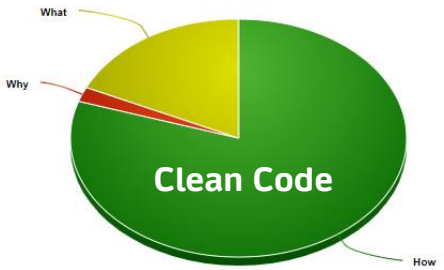
Colloquium on
Clean Code Practices

MTU | LERO
3rd of June 2021

Obectives

- Process towards clean code ✓
- Main components of clean code with some examples ✓
- PEP8 in Depth ✗
- Tools in depth ✗
- P vs. NP ✗

This talk is about..

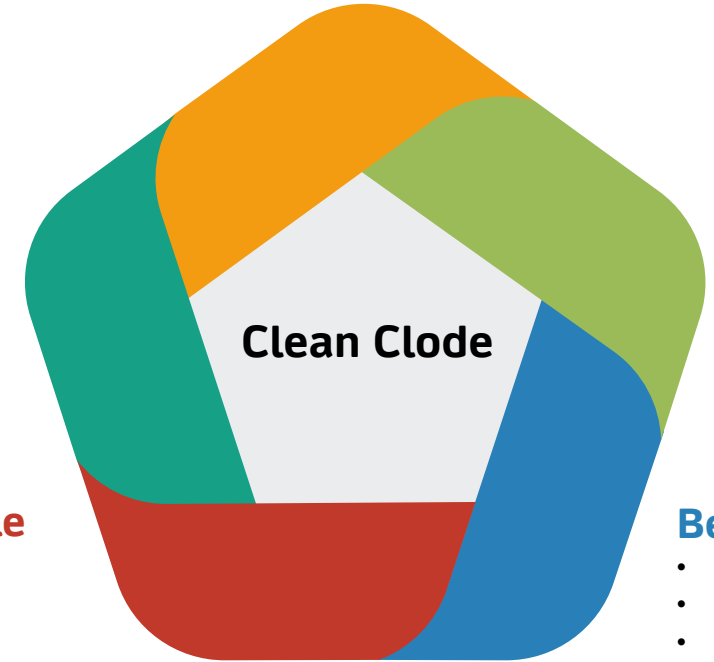


Contents

CLEAN CODE CHARACTERISTICS

Easy to Maintain

- Simple
- Well documented
- Easy to manage



Efficient

- Data structures
- Algorithms

Coding Standards

- Coding/documentation consistency
- Style guides
- Documentation standards
- Linting

Reproducible

- READMEs
- Environments
- Containers
- Documentation
- Unit Tests

Best Practices

- Design patterns
- Industrial conventions
- Language specific constructs (e.g Pythonic Code)

C++ Core Guidelines

March 11, 2021

Editors:

- Bjarne Stroustrup
- Herb Sutter

This is a living document under continuous improvement. Had it been an open-sour
Copying, use, modification, and creation of derivative works from this project is licer
requires agreeing to a Contributor License. See the accompanying LICENSE file for
use, copy, modify, and derive from, hoping for constructive input.

Comments and suggestions for improvements are most welcome. We plan to mo
and the language and the set of available libraries improve. When commenting,
approach. The list of contributors is here.

Problems:

- The sets of rules have not been completely checked for completeness, co
- Triple question marks (???) mark known missing information
- Update reference sections; many pre-C++11 sources are too old.
- For a more-or-less up-to-date to-do list see: To-do: Unclassified proto-

You can read an explanation of the scope and structure of this Guide or jus

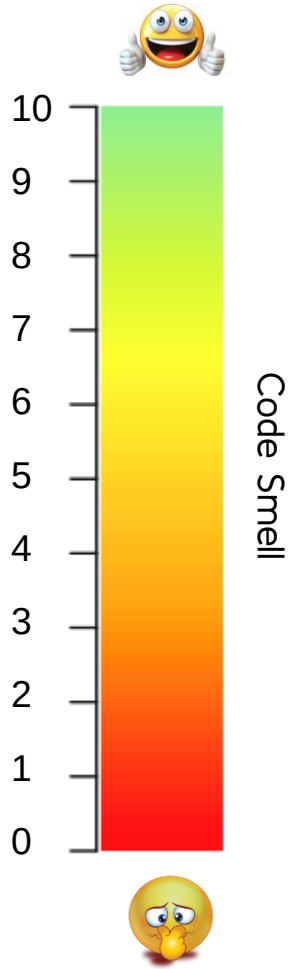
- In: Introduction
- P: Philosophy
- I: Interfaces
- F: Functions
- C: Classes and class hierarchies
- Enum: Enumerations
- R: Resource management
- ES: Expressions and statements

STYLE GUIDES

PEP 8 – Style Guide for Python

“
- Imports should usually be on separate lines
- Use inline comments sparingly
- Conventions for writing good documentation strings (a.k.a. "docstrings") are immortalized in PEP 257.
- Never use the characters 'l' , 'O’ as variable names. ”

pep8 cheat sheet



CODE LINTING

> “lint, or a linter, is a static code analysis tool used to flag programming errors, bugs, stylistic errors and suspicious constructs”

```
1  #include <string>
2  #include <vector>
3
4  int main(int argc, char *argv[]) {
5      int i;
6      std::vector<std::string> v;
7
8      v.puch_back("test");
9
10     return 0;
11 }
```

```
demo.cpp:
    5:9    warning        gcc: warning      unused variable 'i' [-Wunused-variable]
    8:7    error          gcc: error        'class std::vector<std::__cxx11::basic_str
```

https://github.com/SublimeLinter/SublimeLinter-gcc

Enough talking!

Lets write some code.

Demo

Task: Implement Vaccine Registration Module

Acceptance Criteria:

Module MUST:

- provide a method for registration

  Accept Fields: NAME, PPSN, AGE

- provide a method for updating available number of vaccine doses

  SET AVAILIABLE DOSES = 12313

- provide a method to retrieve batches of candidates to notify at any given time. prioritise oldest citizens when retrieving candidates for notification

Possible Sequence of Actions

- set_doses(3) # Authority sets the available number of vaccines
- # Opens web for registration
- add_profile("John Doyle", "4521F", 87)
- add_profile("Liam Murphy", "1221E", 101)
- add_profile("Deirdre Doherty", "7271L", 94)
- add_profile("Paul Ryan", "4531E", 84)
- # Authority decides to notify people
- notify() → we have 3 doses -> should notify 3 oldest persons
- # More registrations
- add_profile("Aisling Gallagher", "5620D", 103)
- # More vaccines coming in
- set_doses(7)
- notify() -> notify remaining oldest people

Liam Murphy

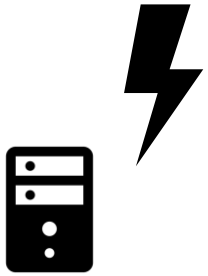
Deirdre Doherty

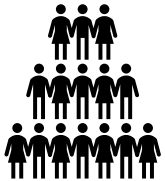
John Doyle

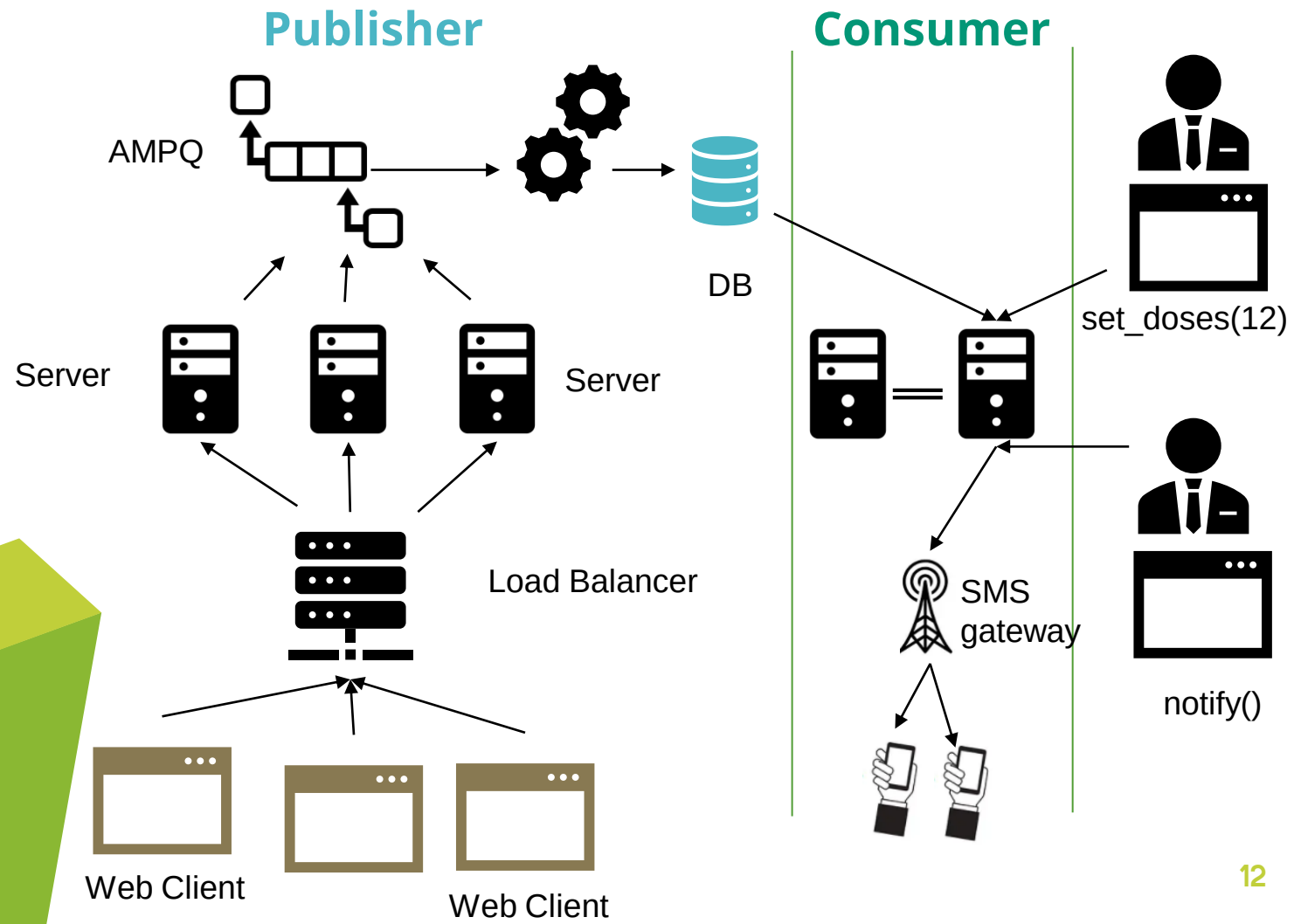
Aisling Gallagher

Paul Ryan

PEP8
Linting
DocStrings
Data Structures






Publisher
Consumer
AMPQ
DB
Server
Server
Load Balancer
SMS
gateway
set_doses(12)
notify()
Web Client
Web Client

“Code tells you how;
Comments tell you why.”

— *Jeff Atwood*

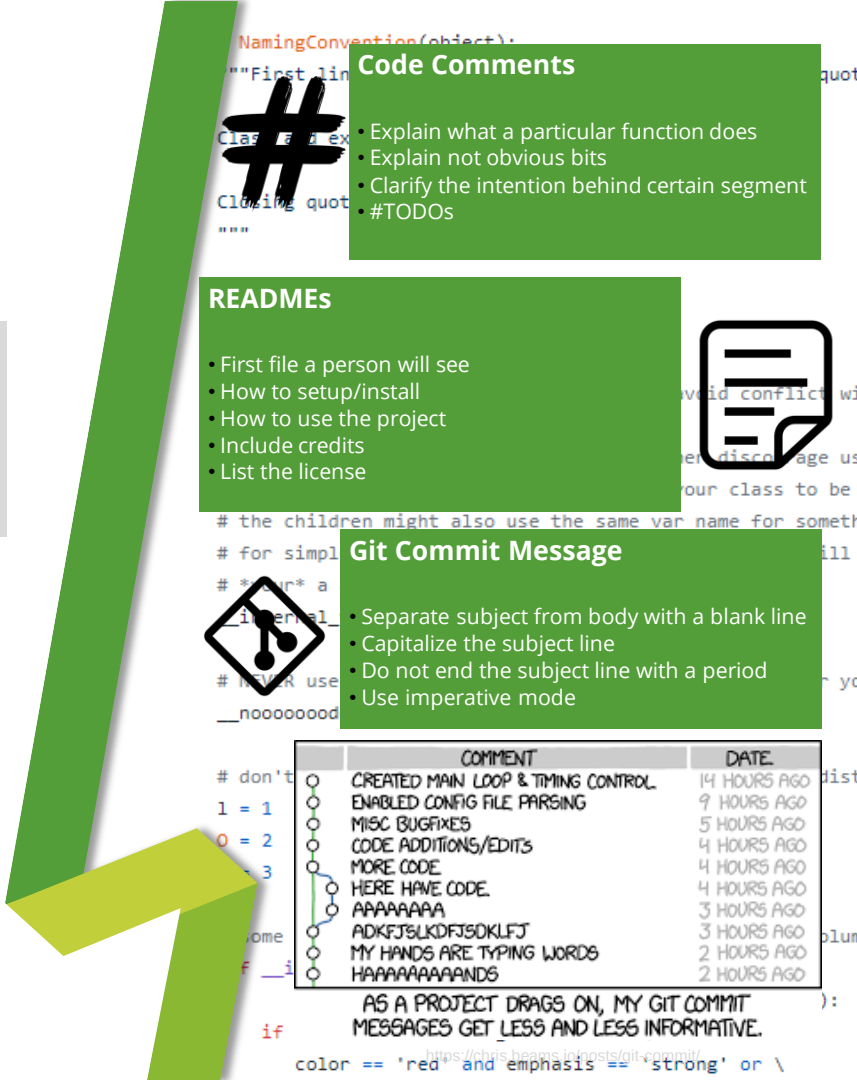
stackoverflow

DOCUMENTATION

reST, Epytext, Google..

> “a *docstring* is a string literal specified in source code that is used, like a comment, to document a specific segment of code”

- Code Comments
- READMEs
- Git Commits
- User/Installation Guides



CLEAN CODE ≠ ONE LINERS

(All the time)

```
mystery_function(N): lambda N: reduce( (lambda r, x: r - set(range(x**2,N,x))
                     if (x in r) else r), range(2,N), set(range(2,N)))
```

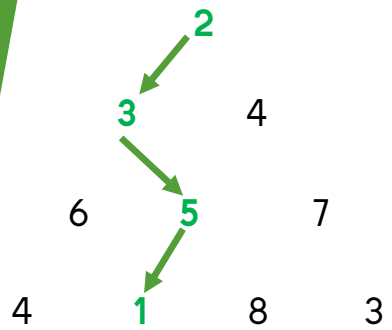
```
def minimumTotal(self, t):
    return reduce(lambda a,b:[f+min(d,e)for d,e,f in zip(a,a[1:],b)],t[::-1])[0]
```

```
int minimumTotal(vector<vector<int>>&$) {
    f(_,$$($)-2,0,1)f(__,_,0,1)$[_][__]+=min($[_+1][__],$[_+1][__+1]);_($[0][0])
}
```

[redacted] ★ 32 June 27, 2015 1:21 AM

This literally makes no sense.

Do you mind expand it and provide an explanation of what is going on?



Minimum Path Sum
2+3+5+1 = 11

6 STEPS TO CLEAN CODE

Assess

Coding standards suitable for your organization/work

Setup a standards commitee

Set goals for the team

Monitor progress

Educate

Team about standards & best practices

Organize brown-bag sessions, trainings, maintain blogs

Identify Tools

That can facilitate implementing coding standards

IDEs, Pylint, PyTests, Flake8

Enforce

Through strict code reviews

Strict rules for linting and unit tests

Through Continuous Integration, Continuous Delivery Pipelines

Refactor

Code as a planned sprint activity

Set refactoring goals for each sprint

Create Templates

Cookie-cutter templates, Python Notebooks, Python Packages

Obectives

- Process towards clean code 
- Main components of clean code with some examples 

Contents

Thank You.