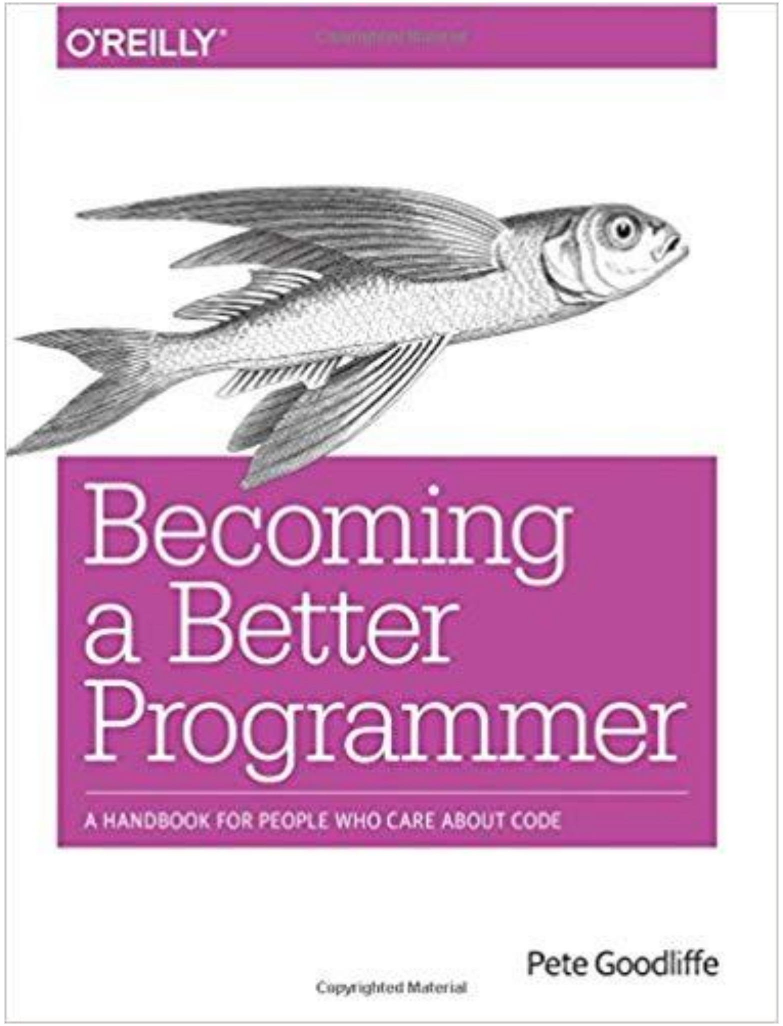
Software Best Practices

Alex Olivas
Madison Bootcamp 2021

https://amzn.to/2I1YydH


O'REILLY
Copyrighted Material
Becoming a Better Programmer
A HANDBOOK FOR PEOPLE WHO CARE ABOUT CODE
Pete Goodliffe
Copyrighted Material

People Power

If you want to go fast, go alone.
If you want to far, go together.

Good software development isn't technical as much as it is social.

- Readability Counts - Zen of Python
- Optimize for the reader, not the writer - Google Style Guide
- "Code is meant to be readable to other programmers and only incidentally to be run on machines" - Paraphrased from ???
- "Functions should fit on a screen and be simple enough that a competent teenager can understand it." - Linux Kernel Style Guide

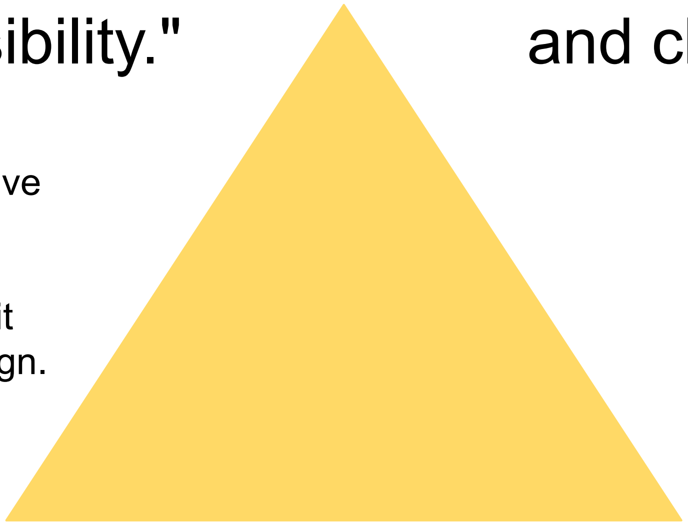
Golden Triangle

In search of a new name!

"Give one entity one cohesive responsibility."

Code

"Correctness, simplicity, and clarity come first."



Entities with one true cohesive responsibility is easy to test.

If an entity is difficult to test it might be a sign of poor design.

Simple code is easy to test and document.

Don't leave to documentation what the compiler or runtime system can enforce.

Tests

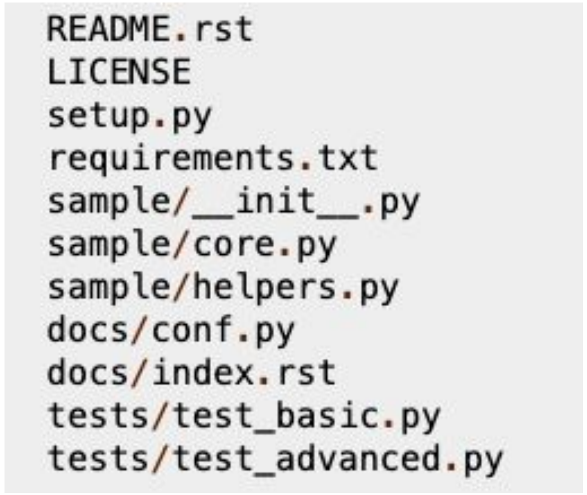
I've never regretted writing tests or documentation. I've sometimes regretted not writing tests or docs..

Documentation

Effective Version Control

Repository Layout

Invest time in a clear, easy to navigate layout.



```
README.rst
LICENSE
setup.py
requirements.txt
sample/__init__.py
sample/core.py
sample/helpers.py
docs/conf.py
docs/index.rst
tests/test_basic.py
tests/test_advanced.py
```

Prefer a standard layout to make it easy for others to navigate your project. Increases adoption.

For python projects prefer virtual environments

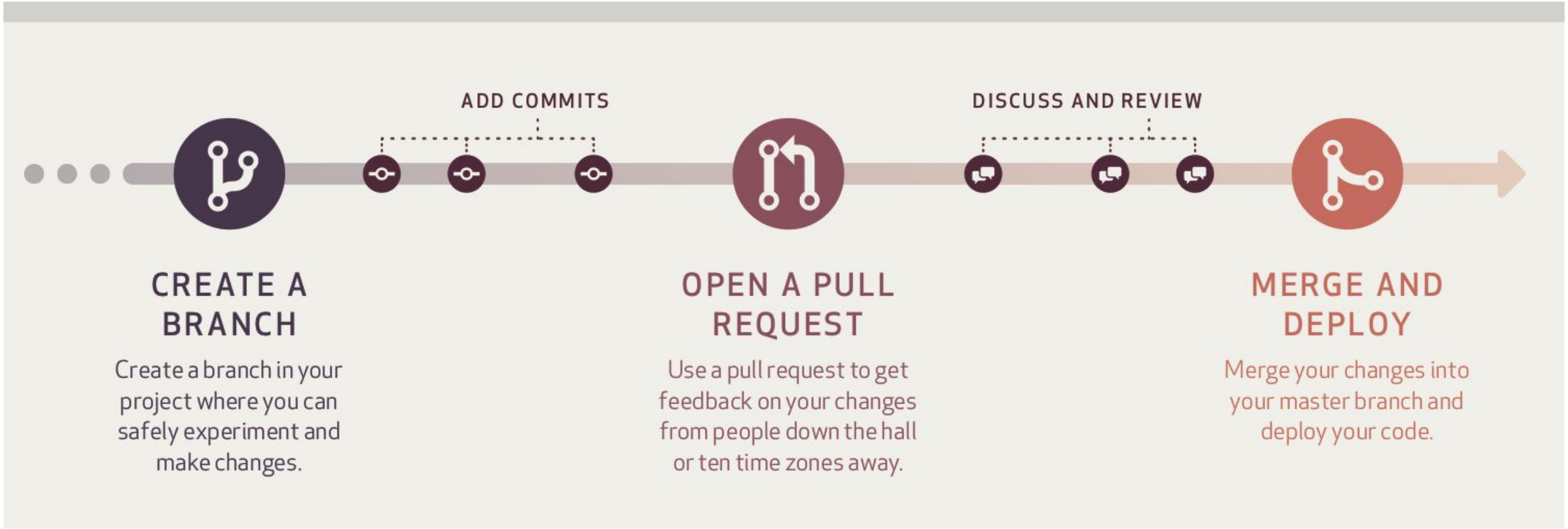
```
$ python3 -m venv my_virtual_env
$ source my_virtual_venv
$ pip install <some_lib>
$ pip freeze > requirements.txt
$ pip install -r requirements.txt
```

https://docs.python-guide.org/writing/structure/

Effective Version Control

1. **Prefer short-lived branches (i.e. days)**
2. **Make small atomic commits.**
3. **Don't break the build.**
4. **Don't mix whitespace/formatting commits with functional changes.**

https://guides.github.com/introduction/flow/


ADD COMMITS
DISCUSS AND REVIEW
CREATE A BRANCH
Create a branch in your project where you can safely experiment and make changes.
OPEN A PULL REQUEST
Use a pull request to get feedback on your changes from people down the hall or ten time zones away.
MERGE AND DEPLOY
Merge your changes into your master branch and deploy your code.

Effective Version Control

Make Small, Atomic Commits
"Commit early, commit often."

- Make the commits small enough that they don't break the code. What constitutes "broken" code? - Doesn't compile. Tests don't pass.
- **DO NOT** commit something that covers more than one change: "git commit -m 'Refactor and critical bugfix and new feature and reformatted.' "
- **DO NOT** wait until the end of the day or week to commit.
- **DO NOT** mix functional changes with whitespace cleanups.
- **DO** write good commit messages.
  - Good commit message: "Fixes issue #123: Use std::shared_ptr to avoid memory leaks. See C++ Coding Standards for more information."
  - Bad commit message: "blerg"

Effective Version Control

Make Quality Commits - **Don't break the build!!!**

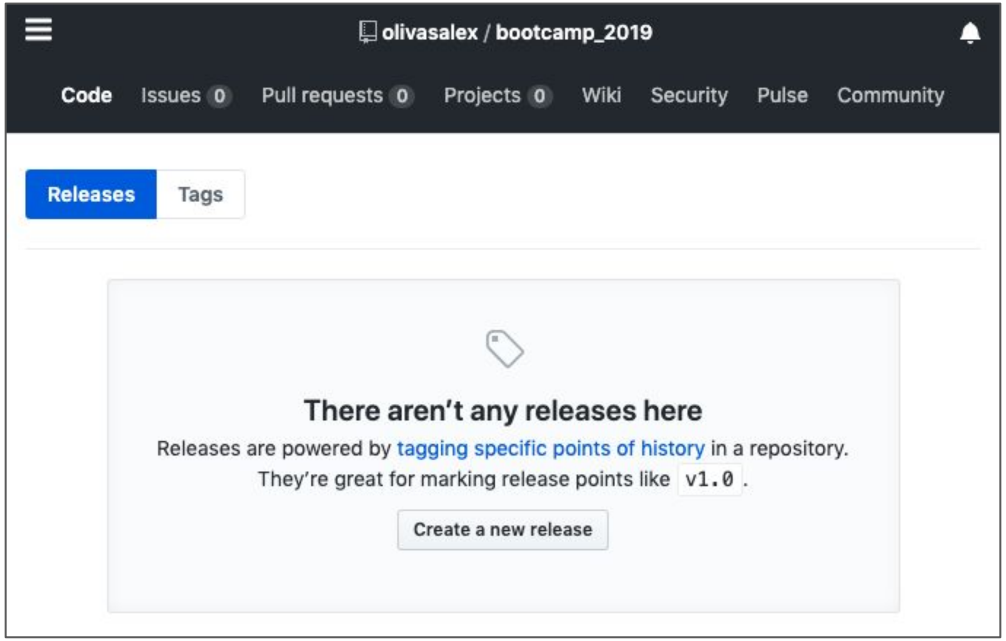
1. Make change
2. Test that it builds against main
3. Ensure all the tests pass. **(Invest in a test suite!!! e.g. python unittest)**
4. Check it in w/ an **informative** commit message.
5. Check your continuous integration (CI) system.

Measure test coverage with pycoverge and display this in README.md

Effective Version Control

Making Releases - https://help.github.com/en/articles/creating-releases

Generating DOIs - https://guides.github.com/activities/citable-code/

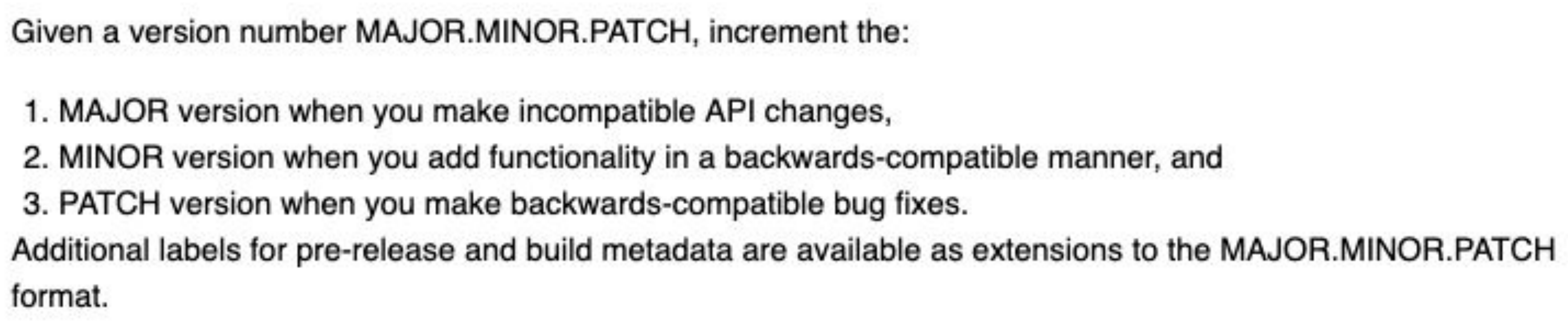


There aren't any releases here

Releases are powered by tagging specific points of history in a repository. They're great for marking release points like `v1.0`.

Effective Version Control

Semantic Version Numbers - https://semver.org/



Given a version number MAJOR.MINOR.PATCH, increment the:

1. MAJOR version when you make incompatible API changes,
2. MINOR version when you add functionality in a backwards-compatible manner, and
3. PATCH version when you make backwards-compatible bug fixes.

Additional labels for pre-release and build metadata are available as extensions to the MAJOR.MINOR.PATCH format.

Coding Standards and Style Guides

Choose a style guide and integrate a linter into your workflow

- Python PEP8 - https://www.python.org/dev/peps/pep-0008/
- Google Style Guides (14) - https://google.github.io/styleguide/
- C++ - https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines

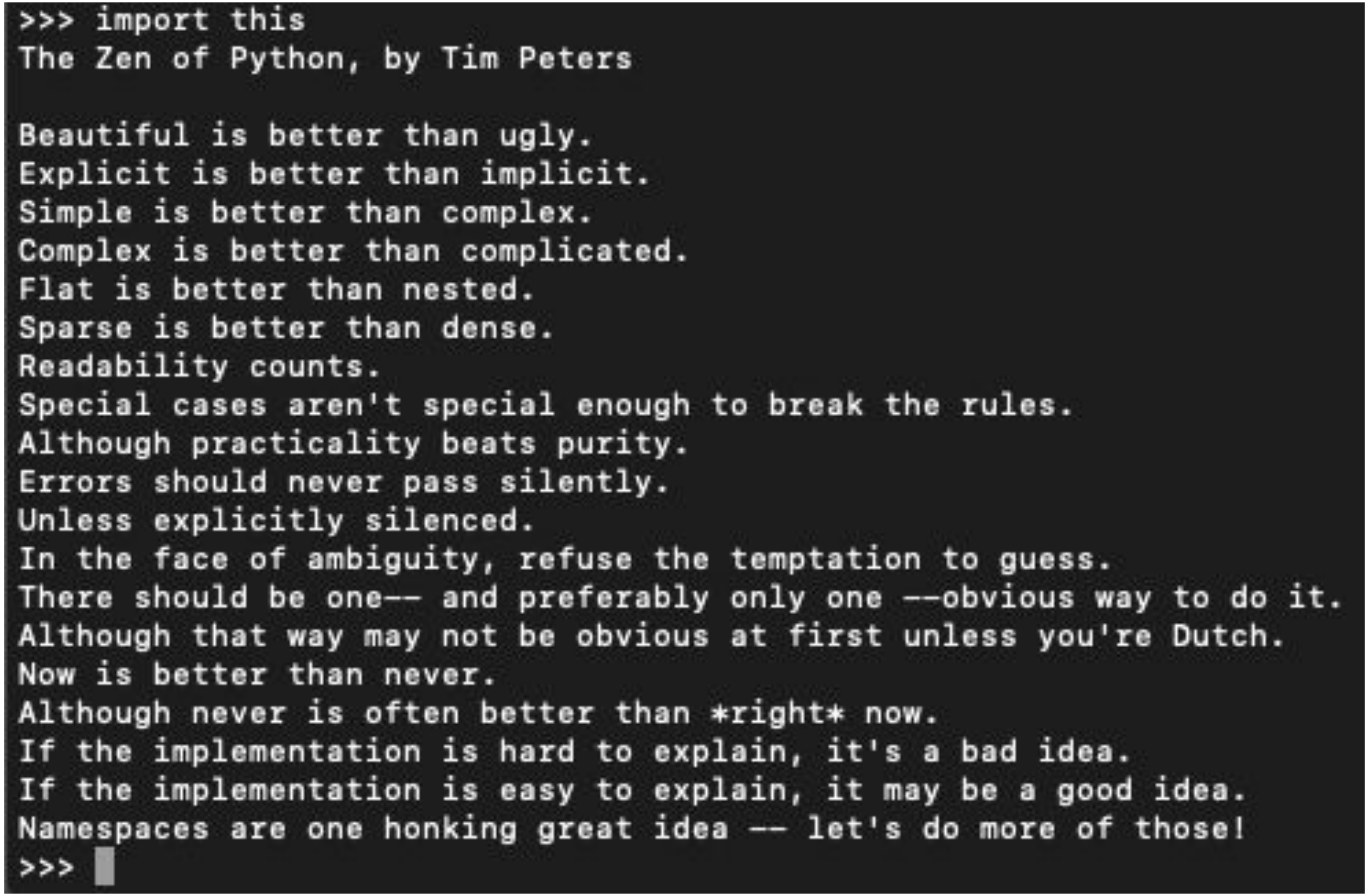
Linters - Code Checkers

- C++
  - cpplint - https://github.com/cpplint/cpplint
  - clang-analyzer - https://clang-analyzer.llvm.org
- Python
  - flake8 - https://flake8.pycqa.org
  - black - https://black.readthedocs.io

IceCube Software Standing Orders

1. Be nice to people.
2. Give one entity one cohesive responsibility.
3. Correctness, **simplicity**, and **clarity** come first.
4. Use tests and documentation as internal design checks.
5. Strive for high test coverage.
6. Make small, atomic commits.
7. Keep whitespace commits separate from functional changes.
8. Prefer short-lived branches.
9. Don't break the build!
10. Invest in and commit to coding standards.

When in doubt “import this”

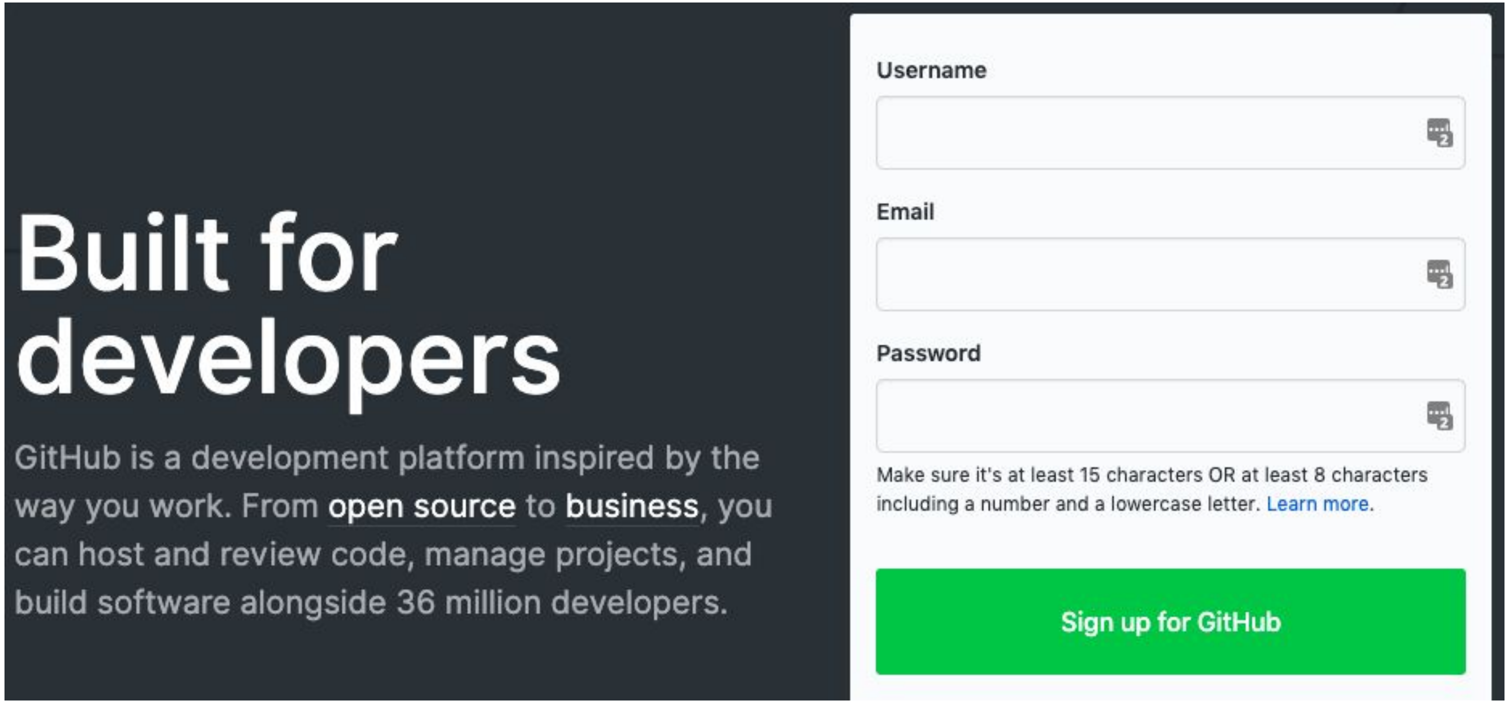


```
>>> import this
The Zen of Python, by Tim Peters

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!
>>> 
```

Exercise: Getting Started w/ GitHub

Use it or Lose it! https://github.com/



Built for developers

GitHub is a development platform inspired by the way you work. From open source to business, you can host and review code, manage projects, and build software alongside 36 million developers.

Username

Email

Password

Make sure it's at least 15 characters OR at least 8 characters including a number and a lowercase letter. Learn more.