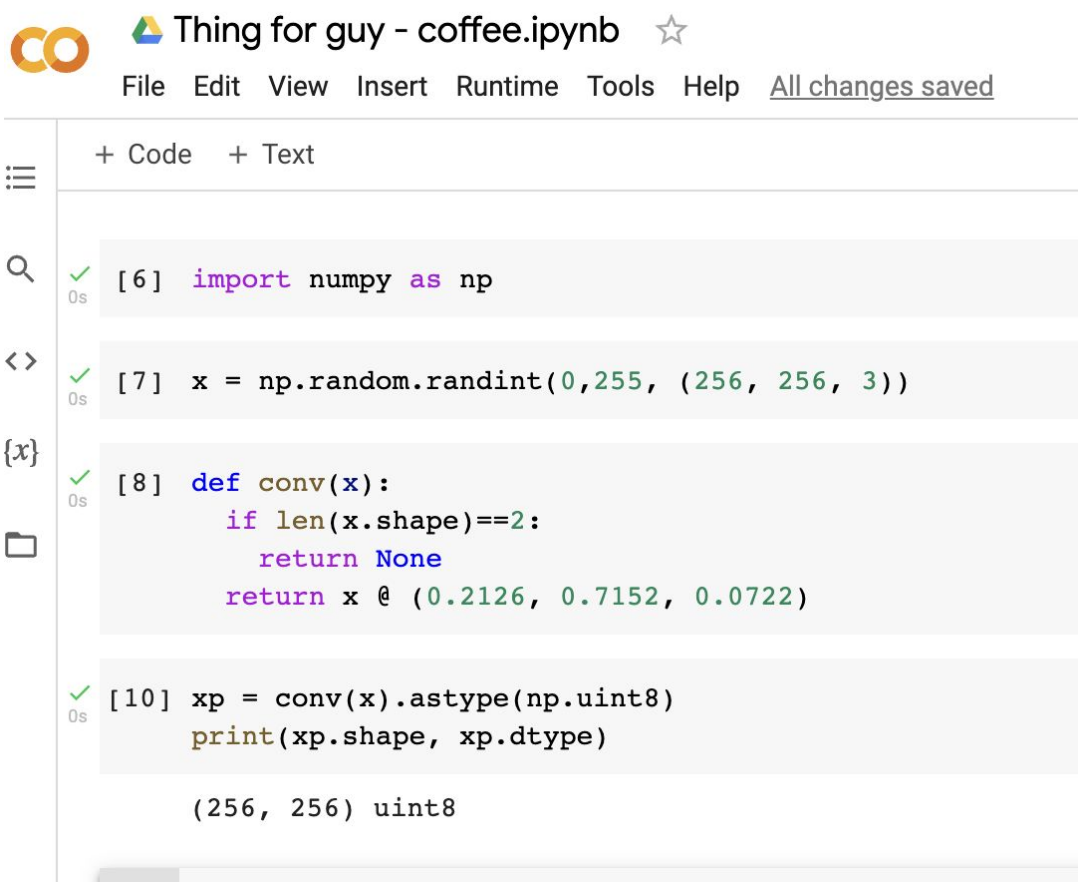
Data Science Survival Skills

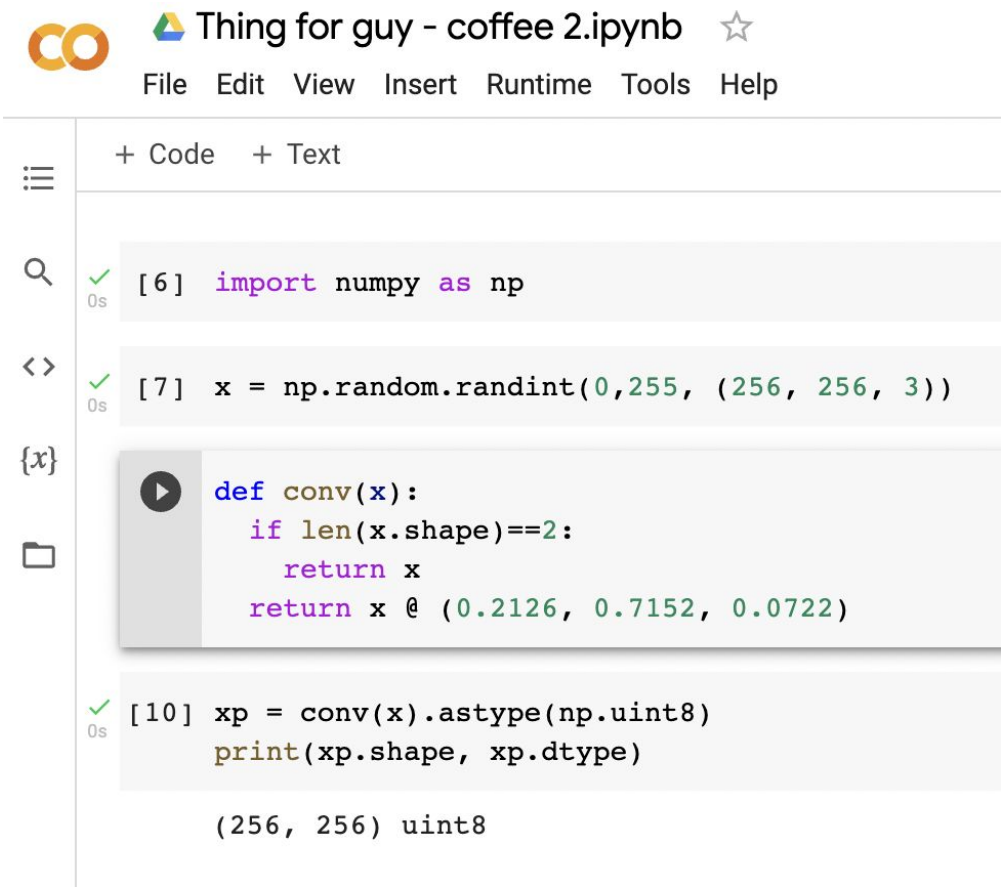
Version Control and Python Package Management

“Sure, I just need 10 lines of code...”

Here you go!


Thing for guy - coffee.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
[6] import numpy as np
0s
[7] x = np.random.randint(0,255, (256, 256, 3))
0s
[8] def conv(x):
if len(x.shape)==2:
return None
return x @ (0.2126, 0.7152, 0.0722)
0s
[10] xp = conv(x).astype(np.uint8)
print(xp.shape, xp.dtype)
0s
(256, 256) uint8

Here you go!!


Thing for guy - coffee 2.ipynb
File Edit View Insert Runtime Tools Help
+ Code + Text
[6] import numpy as np
0s
[7] x = np.random.randint(0,255, (256, 256, 3))
0s
def conv(x):
if len(x.shape)==2:
return x
return x @ (0.2126, 0.7152, 0.0722)
[10] xp = conv(x).astype(np.uint8)
print(xp.shape, xp.dtype)
0s
(256, 256) uint8

Versioning

- Thesis.docx
- Thesis_1.docx
- Thesis_1_anki.docx
- Thesis_2.docx
- Thesis_2_anki.docx
- Thesis_2_AB.docx
- .....
- Thesis_final.docx
- Thesis_final_anki.docx
- Thesis_final2.docx
- Thesis_final2_fix.docx

?!?

Why versioning?

Versioning helps you keep tracking changes

Versioning allows collaboration

Versioning allows rolling back changes

Versioning helps with code review and improves project management

Software versioning

Semantic versioning

Example: Python versioning, e.g. 2.7 and 3.10 ➜ major change may indicate incompatibilities and breaking changes!

More on SemVar

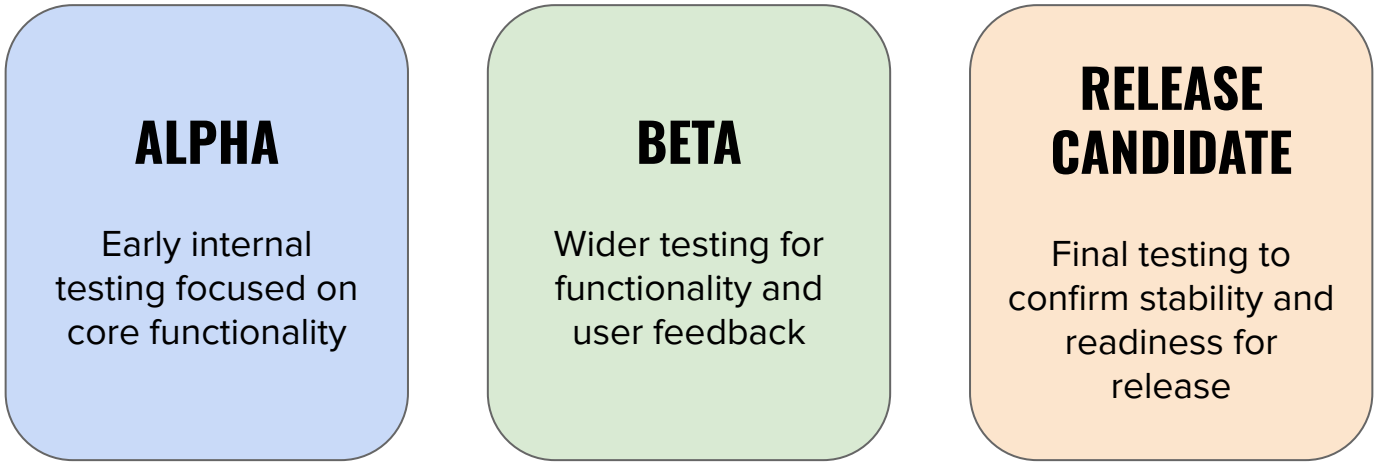
Comparison of development stage indicators

Stage	Semver	Num. Status	Num 90+
Alpha	1.2.0-a.1	1.2.0.1	1.1.90
Beta	1.2.0-b.2	1.2.1.2	1.1.93
Release candidate	1.2.0-rc.3	1.2.2.3	1.1.97
Release	1.2.0	1.2.3.0	1.2.0
Post-release fixes	1.2.5	1.2.3.5	1.2.5

Software Development Lifecycle Stages

Purpose of Staging

- Staging allows controlled testing, bug fixing, and feedback collection.
- Each stage has distinct goals for stability, functionality, and usability before a full release.



Alpha and Beta

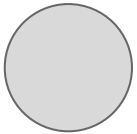
ALPHA

- Internally tested
- Not feature complete
- Serious performance issues
- (un)known bugs
- Software may crash often

BETA

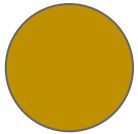
- Feature complete
- Contains significant less bugs
- Still performance, speed issues
- Can also crash and data loss may occur
- Interaction with users ➜ usability testing

Release Candidate and Release



Release Candidate (SILVER)

- Final beta product with acceptable bugs
- Minimal interaction with source code and documentation



Release (GOLD)

- Release to Manufacturing (RTM)
- **Digitally signed**
  ➜ knowing product state

Code readability

Code readability counts

The ease with which a programmer can understand and interpret code.

- Characteristics of Readable Code:
  - Clear logic and structure.
  - Meaningful variable and function names.
  - Consistent coding style.
  - **Well-placed comments explaining the 'why'.**

- Primary Goal: Make the codebase **maintainable**.

- Benefits:
  - Easier modification and extension of code.
  - Reduction in errors.
  - Time-saving for developers.
  - Facilitates collaboration.

Code is read more often than it's written.

Variable and Function names

Variables:

1. Poor: `a`
   Good: `totalAmount`
2. Poor: `str`
   Good: `customerName`
3. Poor: `flag`
   Good: `isUserAuthenticated`
4. Poor: `temp`
   Good: `temporaryFilePath`
5. Poor: `lst`
   Good: `productList`

Functions:

1. Poor: `proc()`
   Good: `processData()`
2. Poor: `uf()`
   Good: `updateFile()`
3. Poor: `cv()`
   Good: `calculateVolume()`
4. Poor: `gi()`
   Good: `getUserInput()`
5. Poor: `rd()`
   Good: `retrieveData()`

Avoid abbreviations, use descriptive names, follow naming conventions.

Naming conventions

Multiple-word identifier formats

Formatting	Name(s)
`twowords`	flatcase[13][14]
`TWOWORDS`	UPPERCASE, SCREAMINGCAMELCASE[13]
`twoWords`	(lower) camelCase, dromedaryCase
`TwoWords`	PascalCase, UpperCamelCase, StudlyCase[15]
`two_words`	snake_case, snail_case, pothole_case
`TWO_WORDS`	ALL_CAPS, SCREAMING_SNAKE_CASE,[16] MACRO_CASE, CONSTANT_CASE
`two_Words`	camel_Snake_Case
`Two_Words`	Pascal_Snake_Case, Title_Case
`two-words`	kebab-case, dash-case, lisp-case, spinal-case
`TWO-WORDS`	TRAIN-CASE, COBOL-CASE, SCREAMING-KEBAB-CASE
`Two-Words`	Train-Case,[13] HTTP-Header-Case[17]

Recommendations:

Python **UpperCamelCase**, private functions with leading underscores (e.g. __init__)

C: lowercase/flatcase
C++: snake_case
C#: camelCase

Hungarian Notation

Systems Hungarian Notation:
The prefix indicates the variable's type.

Apps Hungarian Notation:
The prefix indicates the variable's purpose or logical type, rather than its physical type.

➜ Now controversial, but nice to know for legacy code

Type/Use	Systems Hungarian	Apps Hungarian	Example
Integer	`i`	`count`, `index`, etc.	`iCounter`, `countItems`
Long integer	`l`	`total`	`lTotalBytes`, `totalDuration`
Floating point number	`f`	`ratio`, `percentage`	`fHeight`, `ratioCompletion`
Double precision float	`d`	`average`	`dBalance`, `averageScore`
Character	`c`	`prefix`, `suffix`	`cFirstLetter`, `prefixName`
String	`str`	`name`, `address`, etc.	`strUsername`, `nameFirst`
Boolean	`b`	`is`, `has`, `can`	`bFound`, `isAuthenticated`
Pointer	`p`	-	`pNode`, `pCurrent`
Array	`a`	-	`aValues`, `aNames`
User Interface elements	-	Specific prefixes	`btnSave` (Button), `txtEmail` (Textbox)

Python Enhancement Proposals (PEP)

The Zen of Python

PEP-20

```
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!
```

PEP-07 and PEP-08

- Rule sets for programming C/C++ and Python

"Code is more often read than written."

— *Guido van Rossum*

Readability: 79 characters / line code,
72 characters / line documentation

https://peps.python.org/pep-0008/

	PEP	Title	Authors	
PA	1	PEP Purpose and Guidelines	Barry Warsaw, Jeremy Hylton, David Goodger, Alyssa Coghlan	
PA	2	Procedure for Adding New Modules	Brett Cannon, Martijn Faassen	
PA	4	Deprecation of Standard Modules	Brett Cannon, Martin von Löwis	
PA	7	Style Guide for C Code	Guido van Rossum, Barry Warsaw	
PA	8	Style Guide for Python Code	Guido van Rossum, Barry Warsaw, Alyssa Coghlan	
PA	10	Voting Guidelines	Barry Warsaw	
PA	11	CPython platform support	Martin von Löwis, Brett Cannon	
PA	12	Sample reStructuredText PEP Template	David Goodger, Barry Warsaw, Brett Cannon	
PA	13	Python Language Governance	The Python core team and community	
PA	387	Backwards Compatibility Policy	Benjamin Peterson	
PA	602	Annual Release Cycle for Python	Łukasz Langa	3.9
PA	609	Python Packaging Authority (PyPA) Governance	Dustin Ingram, Pradyun Gedam, Sumana Harihareswara	
PA	676	PEP Infrastructure Process	Adam Turner	
PA	729	Typing governance process	Jelle Zijlstra, Shantanu Jain	
PA	731	C API Working Group Charter	Guido van Rossum, Petr Viktorin, Victor Stinner, Steve Dower, Irit Katriel	
PA	732	The Python Documentation Editorial Board	Joanna Jablonski	

Comments vs Documentation

"Code tells you how; **Comments tell you why.**"

— *Jeff Atwood (aka Coding Horror)*

CODING HORROR
programming and human factors

"It doesn't matter how good your software is, because **if the documentation is not good enough, people will not use it.**"

— *Daniele Procida*

DIVIO

Basics of Commenting Code

```
def hello_world():
    # A simple comment preceding a simple print statement
    print("Hello World")
```

Planning and Reviewing

```
# First step
# Second step
# Third step
```

Code description

```
# Attempt a connection based on previous settings. If unsuccessful,
# prompt user for new settings.
```

Algorithmic description

```
# Using quick sort for performance gains
```

Tagging (TODO, BUG, FIXME,..)

```
# TODO: Add condition for when val is None
```

https://realpython.com/documenting-python-code/

Comment principles

- Have related comments next to the actual code
- Don’t use complex ASCII styles etc, keep it clean
- Don’t be redundant:
  Expect reader to know your language (e.g. Python)
  Don’t comment the obvious (e.g. # use quick sort and next line is lib.quicksort())

- Ideally: Design your code, such that it comments itself
  (and no, most likely you are NOT doing it…)

© Jeff Atwood

Type hinting

```python
from typing import List

def should_use(annotations: List[str]) -> bool:
    print("They're awesome!")
    return True
```

```python
def area_rectangle(length: float, breadth: float) -> float:
    return length * breadth
```

Type hinting != type checking / error raising!

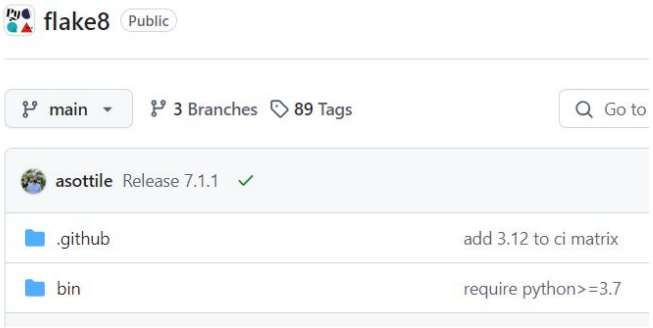
https://dagster.io/blog/python-type-hinting

https://medium.com/depurr/python-type-hinting-a7afe4a5637e

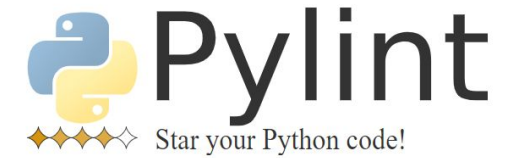
Handling errors w.r.t. type

```python
def my_function(arg1: int, arg2: str):
    # Check types of arguments
    if not isinstance(arg1, int):
        raise TypeError(f"Argument 'arg1' must be of type int, but got {type(arg1).__name
    if not isinstance(arg2, str):
        raise TypeError(f"Argument 'arg2' must be of type str, but got {type(arg2).__name

    # Function logic here
    print("Arguments are of the correct type.")

# Example usage
my_function(5, "example")  # This will work fine
my_function(5, 10)         # This will raise a TypeError
```

Libraries that help you with PEP-8 styles



The Uncompromising Code Formatter



Read the doc | Install it | Contribute | Get support

Coding Standard

- checking line-code's length,
- checking if variable names are well-formed according to your coding standard
- checking if imported modules are used

Python's PEP8 style guide

Docstring

A docstring (PEP 257)

```
def say_hello(name):
    """A simple function that says hello... Richie style"""
      print(f"Hello {name}, is it me you're looking for?")
```

- Class Docstrings: Class and class methods
- Package and Module Docstrings: Package, modules, and functions
- Script Docstrings: Script and functions

```
class SimpleClass:
    """Class docstrings go here."""

    def say_hello(self, name: str):
        """Class method docstrings go here."""

        print(f'Hello {name}')
```

- A brief summary of its purpose and behavior
- Any public methods, along with a brief description
- Any class properties (attributes)
- Anything related to the interface for subclassers, if the class is intended to be subclassed

https://realpython.com/documenting-python-code/

Docstring methods

- A brief description of what the function is and what it’s used for
- Any arguments (both required and optional) that are passed including keyword arguments
- Label any arguments that are considered optional
- Any side effects that occur when executing the function
- Any exceptions that are raised
- Any restrictions on when the function can be called
- Examples

https://realpython.com/documenting-python-code/

Example docstrings

Registration using Phase Cross Correlation:

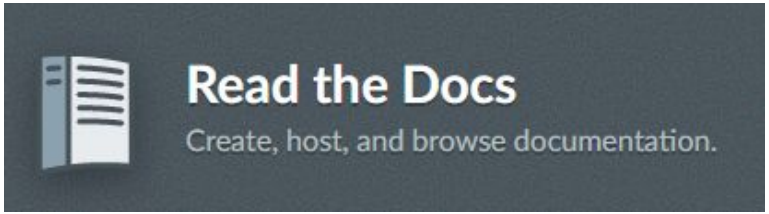
https://github.com/scikit-image/scikit-image/blob/main/skimage/registration/_phase_cross_correlation.py

Drawing an ellipse

https://github.com/scikit-image/scikit-image/blob/main/skimage/draw/draw.py

Documentation

- Sphinx
- Read The Docs



https://www.writethedocs.org/guide/writing/beginners-guide-to-docs/

- You will be using your code in 6 months
- You want people to use your code
- You want people to help out
- You want your code to be better
- You want to be a better writer

How to document a function

Formatting Type	Description	Supported by Sphynx	Formal Specification
Google docstrings	Google's recommended form of documentation	Yes	No
reStructured Text	Official Python documentation standard; Not beginner friendly but feature rich	Yes	Yes
NumPy/SciPy docstrings	NumPy's combination of reStructured and Google Docstrings	Yes	Yes
Epytext	A Python adaptation of Epydoc; Great for Java developers	Not officially	Yes

A function

```python
def factorial(n, double=False):
    if double:
        return 1 if n <= 0 else n * factorial(n - 2, double=True)
    return 1 if n == 0 else n * factorial(n - 1)
```

For a standard factorial calculation, `factorial(10)` should be:

$$10! = 10 \times 9 \times 8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 3,628,800$$

If using the double factorial (with `double=True`), it would calculate:

$$10!! = 10 \times 8 \times 6 \times 4 \times 2 = 3,840$$

What you have learned so far?

```
def factorial(n, double=False):
    if n < 0:
        raise ValueError("n must be a non-negative integer.")
    if double:
        return 1 if n <= 0 else n * factorial(n - 2, double=True)
    return 1 if n == 0 else n * factorial(n - 1)
```

Check bounds

Type hinting

```
def factorial(n: int, double: bool = False) -> int:
    if n < 0:
        raise ValueError("n must be a non-negative integer.")
    if double:
        return 1 if n <= 0 else n * factorial(n - 2, double=True)
    return 1 if n == 0 else n * factorial(n - 1)
```

Google docstring

```python
def factorial(n: int, double: bool = False) -> int:
    """
    Calculates the factorial of a given integer.

    Args:
        n (int): The integer to calculate the factorial of.
        double (bool, optional): If True, calculates the double factorial. Defaults to Fa

    Returns:
        int: The factorial or double factorial of `n`.

    Raises:
        ValueError: If `n` is negative.

    Examples:
        >>> factorial(5)
        120
        >>> factorial(5, double=True)
        15
    """
    # Function logic...
```

reStructered Text docstring

```python
def factorial(n: int, double: bool = False) -> int:
    """
    Calculates the factorial of a given integer.

    :param n: The integer to calculate the factorial of.
    :type n: int
    :param double: If True, calculates the double factorial. Defaults to False.
    :type double: bool, optional
    :return: The factorial or double factorial of `n`.
    :rtype: int
    :raises ValueError: If `n` is negative.

    Example:
        >>> factorial(5)
        120
        >>> factorial(5, double=True)
        15
    """
    # Function logic...
```

Numpy/Scipy docstring

```python
def factorial(n: int, double: bool = False) -> int:
    """
    Calculates the factorial of a given integer.

    Parameters
    ----------
    n : int
        The integer to calculate the factorial of.
    double : bool, optional
        If True, calculates the double factorial. Default is False.

    Returns
    -------
    int
        The factorial or double factorial of `n`.

    Raises
    ------
    ValueError
        If `n` is negative.

    Examples
    --------
    >>> factorial(5)
    120
    >>> factorial(5, double=True)
    15
    """
    # Function logic...
```

What makes a good documentation?

	Most Useful When We're Studying	Most Useful When We're Coding
Practical Step	*Tutorials*	*How-To Guides*
Theoretical Knowledge	*Explanation*	*Reference*

- Tutorials: Lessons that take the reader by the hand through a series of steps to complete a project (or meaningful exercise). Geared towards the user's learning.
- How-To Guides: Guides that take the reader through the steps required to solve a common problem (problem-oriented recipes).
- References: Explanations that clarify and illuminate a particular topic. Geared towards understanding.
- Explanations: Technical descriptions of the machinery and how to operate it (key classes, functions, APIs, and so forth). Think Encyclopedia article.

https://realpython.com/documenting-python-code/

Code Modularity

Code modularity

What is modular programming?

Modular programming is a general programming concept where developers separate program functions into independent pieces. These pieces then act like building blocks, with each block containing all the necessary parts to execute one aspect of functionality.

When the blocks, or modules, are put together, they make up the executable program.

https://www.tiny.cloud/blog/modular-programming-principle/

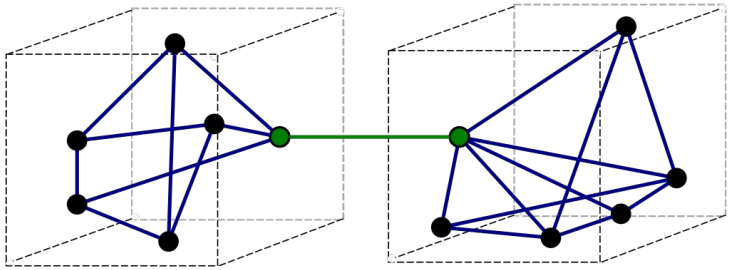
An example (The Single Responsibility Principle.)

```
// Without modularization
function calculatePrice(quantity, price, tax) {
  let subtotal = quantity * price;
  let total = subtotal + (subtotal * tax);
  return total;
}

// Without modularization
let quantity = parseInt(prompt("Enter quantity: "));
let price = parseFloat(prompt("Enter price: "));
let tax = parseFloat(prompt("Enter tax rate: "));

let total = calculatePrice(quantity, price, tax);
console.log("Total: $" + total.toFixed(2));
```

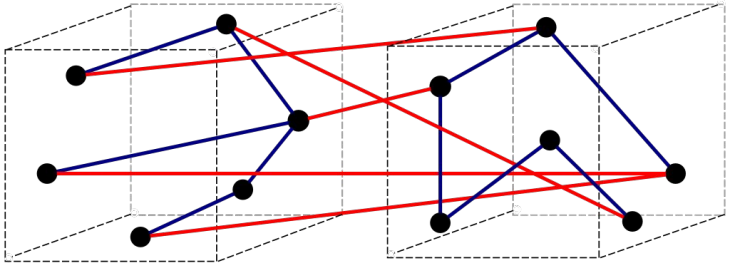
```
// With modularization
function calculateSubtotal(quantity, price) {
  return quantity * price;
}

function calculateTotal(subtotal, tax) {
  return subtotal + (subtotal * tax);
}

// With modularization
let quantity = parseInt(prompt("Enter quantity: "));
let price = parseFloat(prompt("Enter price: "));
let tax = parseFloat(prompt("Enter tax rate: "));

let subtotal = calculateSubtotal(quantity, price);
let total = calculateTotal(subtotal, tax);
console.log("Total: $" + total.toFixed(2));
```

Coupling and cohesion in programming

Coupling refers to the **interdependencies** between modules, while cohesion describes **how related the functions within a single module are**



a) Good (loose coupling, high cohesion)



b) Bad (high coupling, low cohesion)

Error handling

Anticipate errors

Anticipate errors

Y2K BUG

Many computer systems represented years using two digits, leading to potential confusion between 1900 and 2000. Despite extensive testing and remediation, there were concerns about catastrophic failures.

Leap Second BUG

Systems not accounting for the occasional addition of a leap second led to outages in major platforms like Reddit, LinkedIn, and Foursquare in 2012.

Graceful Degradation

The ability **of a system to continue functioning,** albeit at a reduced level, when components fail or when it encounters unforeseen scenarios.

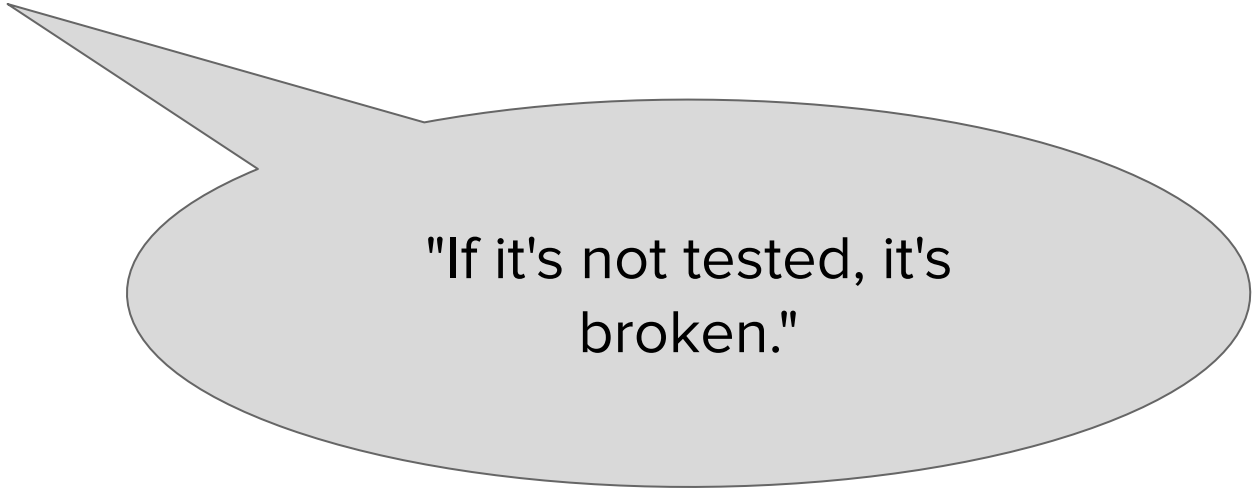
Web Browsers: Websites that still function without JavaScript enabled, but with reduced interactivity or design elements.

Video Streaming: Platforms like YouTube or Netflix reducing video quality in low-bandwidth scenarios instead of stopping playback.

Enhances user trust and satisfaction. Even if not all features are available, the software remains usable.

Testing

Testing is very important



(un)famous examples of not rigorous testing

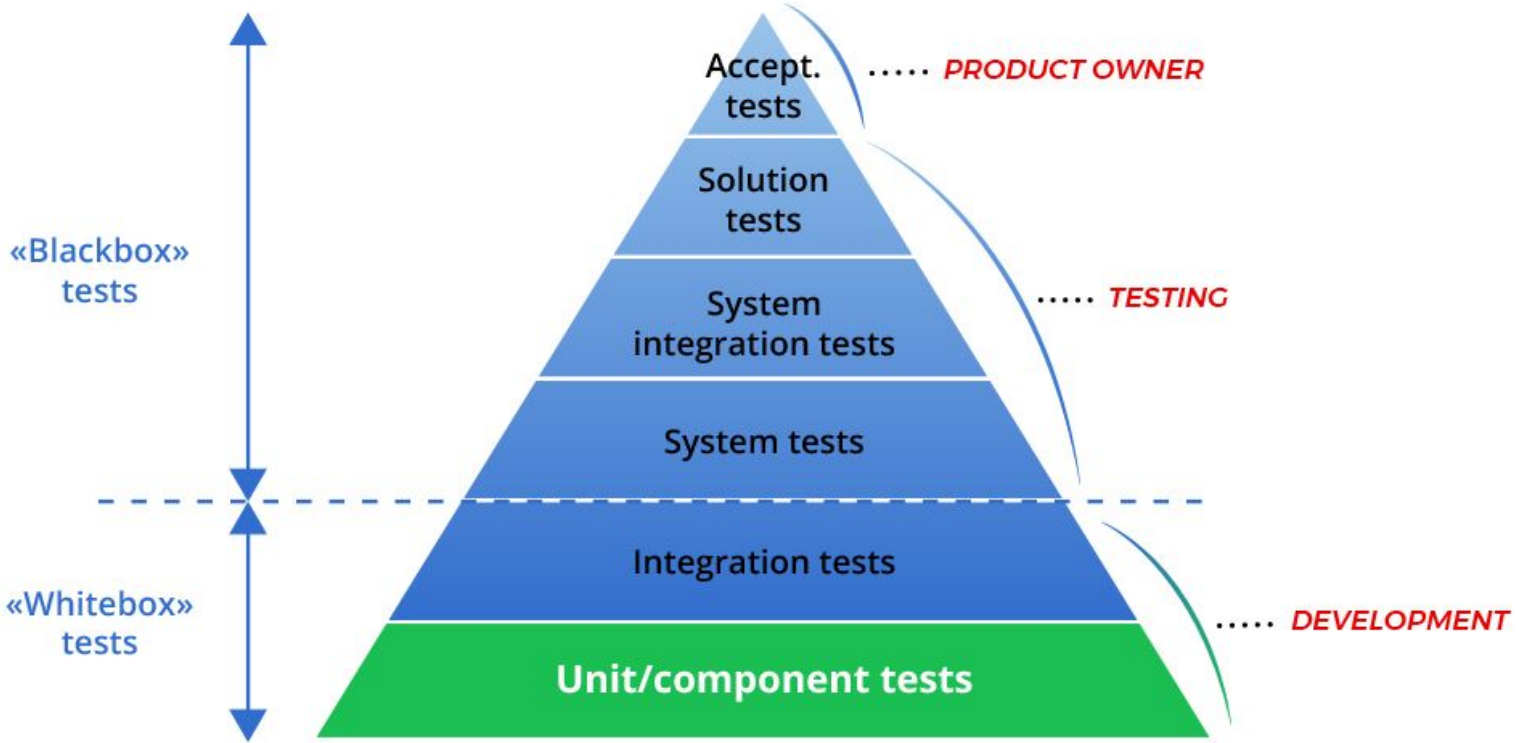
Boeing 737 Max Crashes (2018-2019): Software flaws in the plane's control system led to two fatal crashes.

Knight Capital Group (2012): A software glitch in its trading system caused a $440 million loss in 45 minutes.

Ariane 5 Rocket Explosion (1996): A software error caused a $370 million satellite launch vehicle to self-destruct shortly after liftoff.

Heartbleed Bug (2014): A serious vulnerability in the OpenSSL cryptographic software library, which could have been prevented with rigorous testing.

Unit tests



https://blog.atinternet.com/en/why-invest-in-unit-testing/

Why Unit Tests?

Early Bug Detection: Unit tests help identify bugs at the earliest stages of development, making them less expensive and easier to fix.

Improved Code Quality: Writing tests often results in more modular, maintainable code. It can guide the design of your code (Test-Driven Development).

Refactoring Confidence: With a good set of unit tests, developers can refactor or change code with assurance that any regressions will be caught by the tests.

Documentation: Unit tests serve as documentation by example. They clearly demonstrate how the system should behave under different conditions.

Continuous Integration: Unit tests are essential for implementing Continuous Integration/Continuous Deployment (CI/CD), ensuring that new changes don't break existing functionality.

Collaboration: When multiple developers work on a project, unit tests ensure that one developer's changes don't break another developer's code.

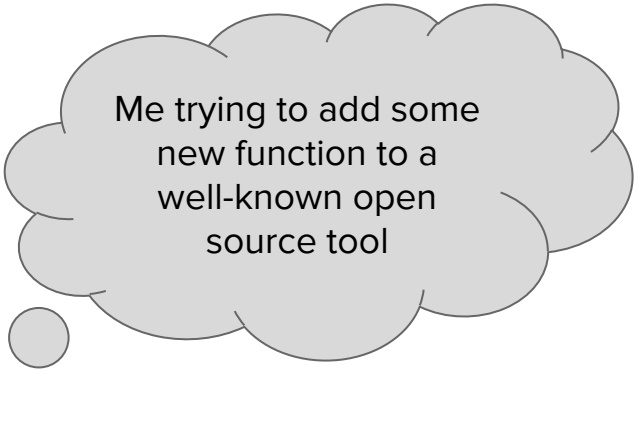
How does a unit test look like?

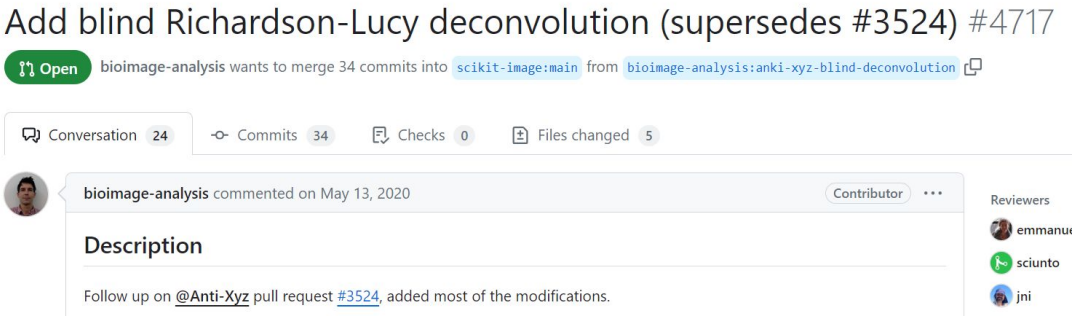
```python
# main_module.py
def add(a, b):
    return a + b

# test_module.py
import unittest
from main_module import add

class TestAddFunction(unittest.TestCase):
    def test_add_positive_numbers(self):
        self.assertEqual(add(3, 2), 5)

    def test_add_negative_numbers(self):
        self.assertEqual(add(-1, -1), -2)

    def test_add_mixed_numbers(self):
        self.assertEqual(add(3, -2), 1)

if __name__ == "__main__":
    unittest.main()
```

We are testing the function “add”.

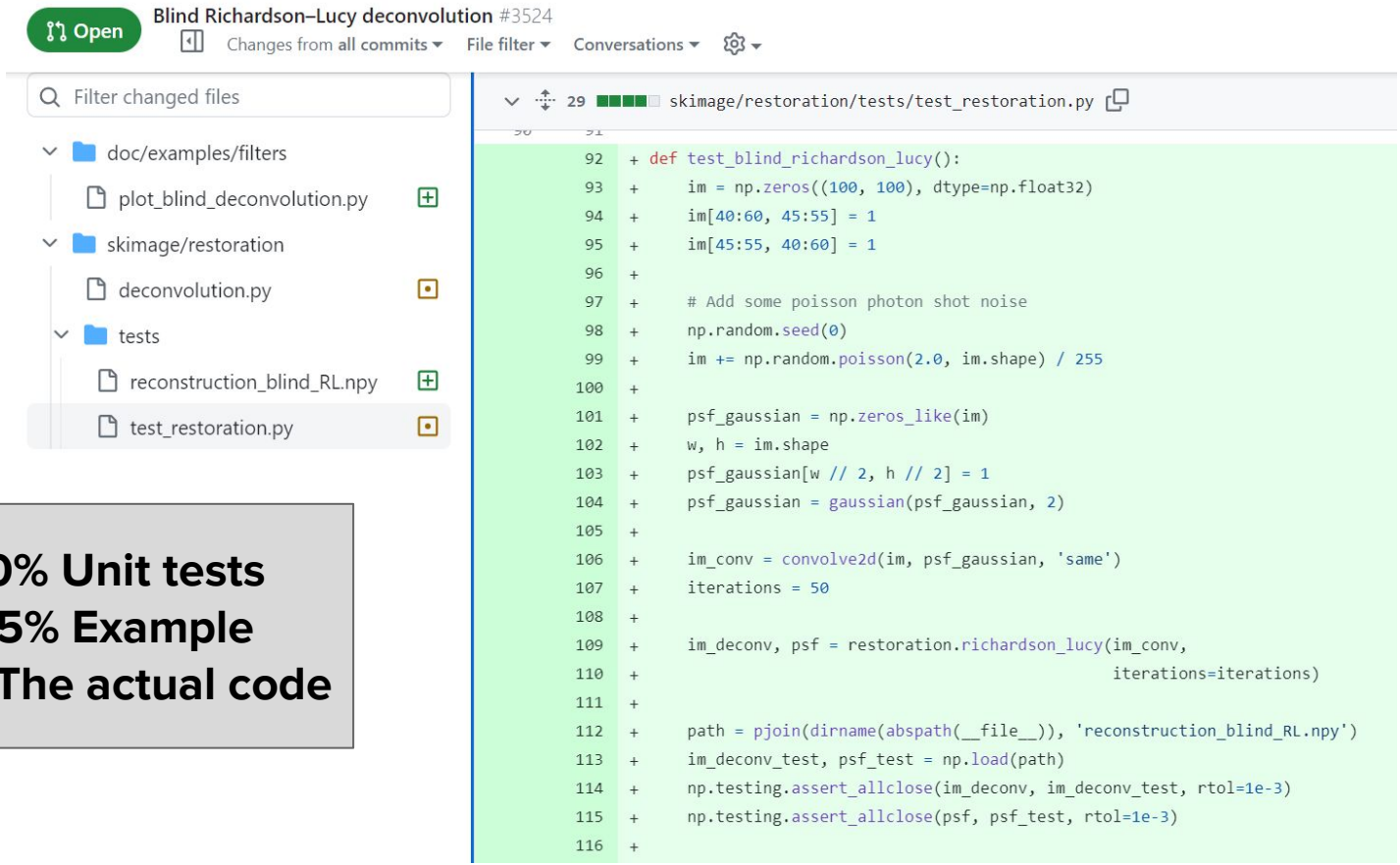
We will verify that
* add behaves correctly by adding two positive numbers
* … two negative numbers
* …. Two mixed numbers

Real-life example





What I wanted to contribute (still pending ~ 5 years)



50% Unit tests
25% Example
25% The actual code

Version control

How can I track more meaningful file versions?

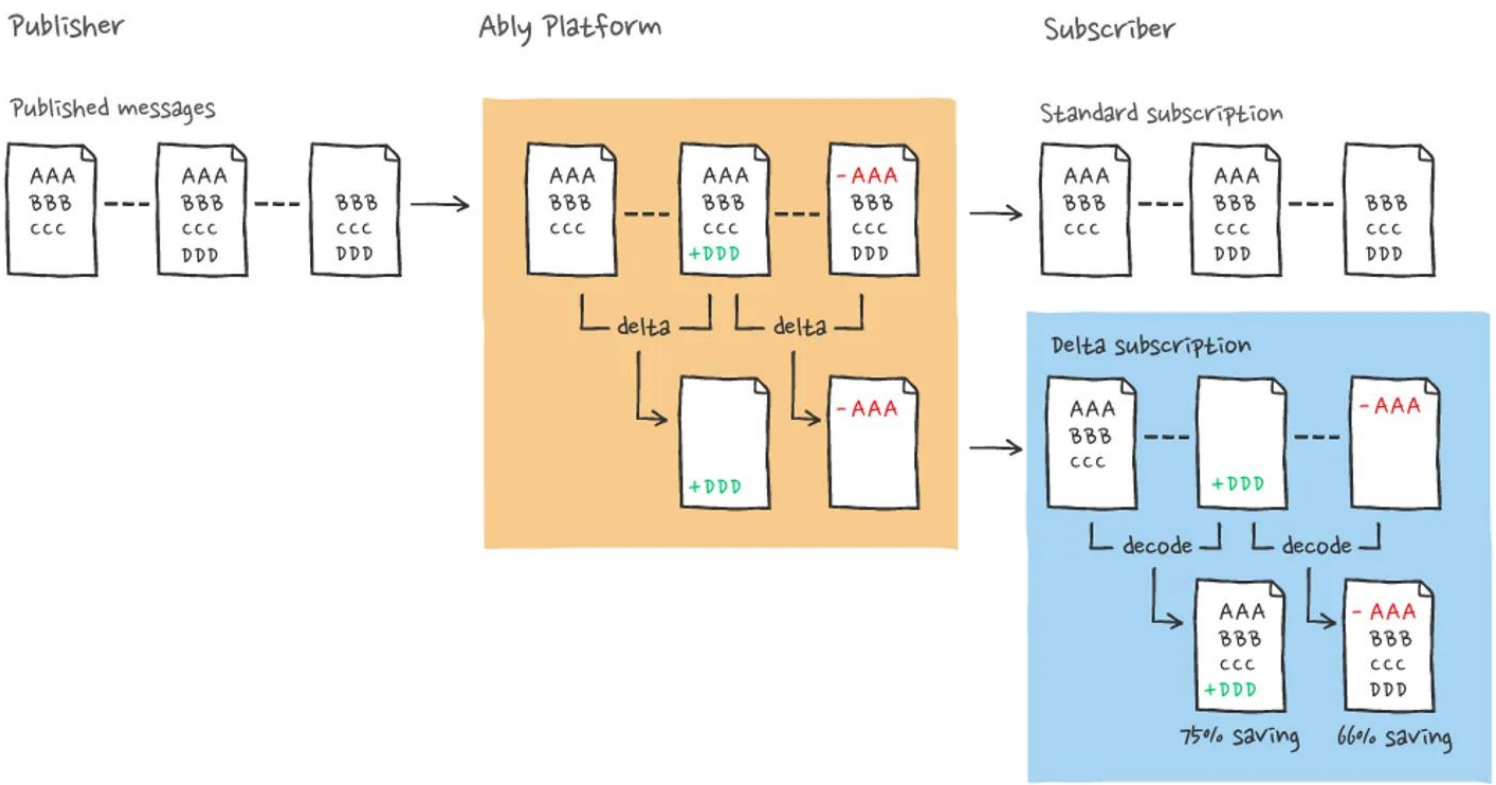
Is there some kind of “version control”?

CVS - Concurrent Versions System

Introduction | News | Documentation | Get the Software | Help and Bug Reports | Development

Already developed in the 80s.
The store changes using **delta compression!**

Delta compression



https://ably.com/blog/practical-guide-to-diff-algorithms

Apache Subversion (SVN)

- Tried to be successor to CVS
- Is used in the following projects:
  Clang, FreeBSD, GCC...
- Fixed a lot of previous bugs in CVS
  And implemented more features

Issue:

- Renaming is copy&delete that is fed back to complete file history ➜ could break things in older versions

The BitKeeper controversy (early 2000s)

BitKeeper: “You can use BitKeeper free of charge for cool freeware and open source project”*

*if you are not actively supporting any competitor to BitKeeper

Used in the Linux kernel by some people...

Andrew Tridgell reverse engineered BitKeeper protocol to create “SourcePuller” -> BitKeeper revoked free licenses.....

Git

https://github.blog/2020-12-17-commits-are-snapshots-not-diffs/

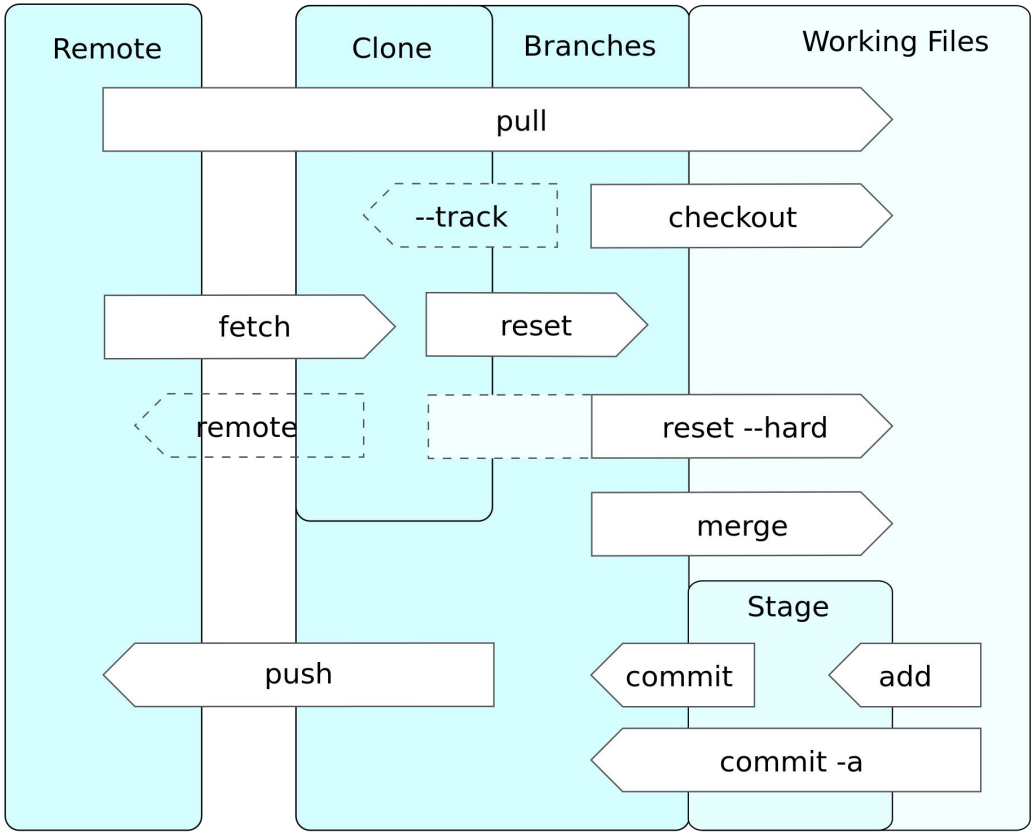
Developed in 1 Month (April 2005)
Powered the Kernel release 2.6.12 release (June 2005)

The Git principle

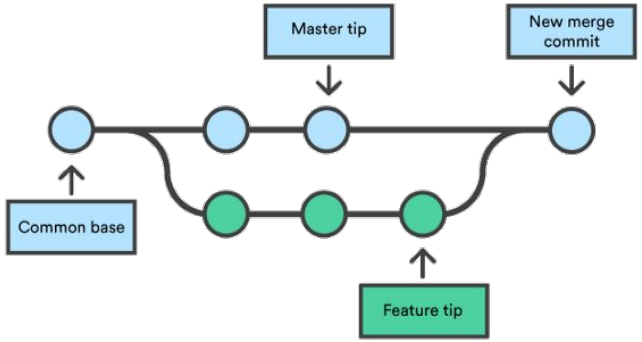
git --everything-is-local

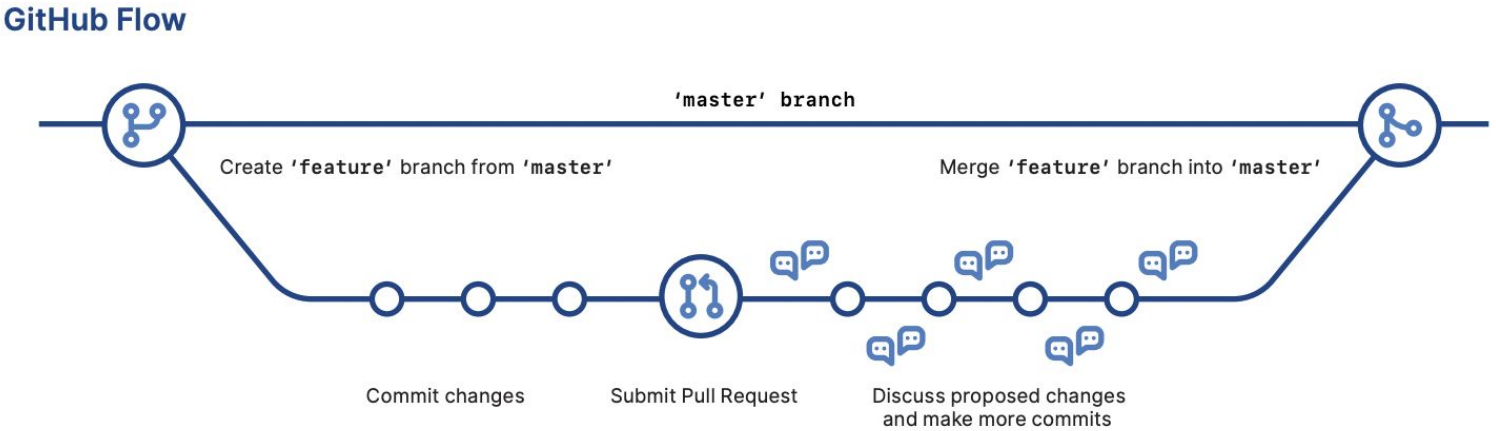
Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

Git is easy to learn and has a tiny footprint with lightning fast performance. It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with features like cheap local branching, convenient staging areas, and multiple workflows.



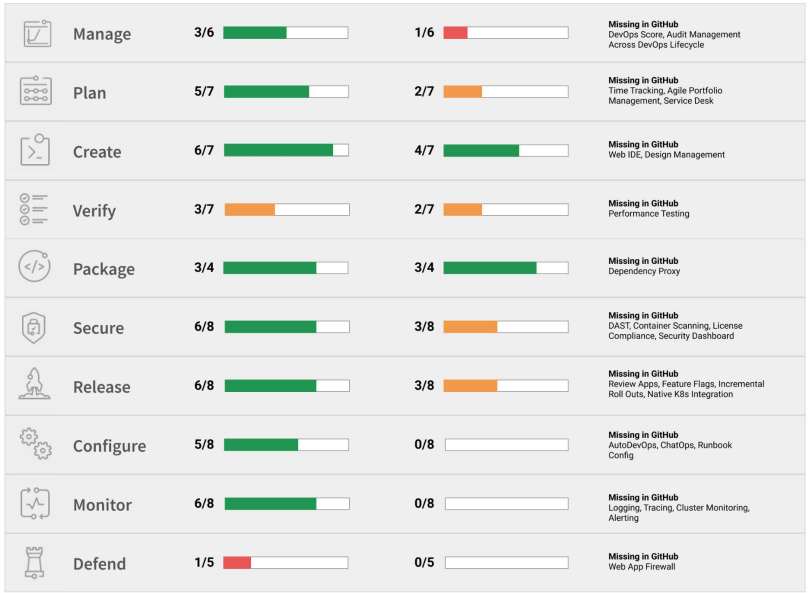
The branching principle





Where to store these “repositories”?

GitLab vs. GitHub Comparison Challenge



Which product is best for you? Share your review on twitter with **#GitChallenge**

Comparing GitLab Terminology

Bitbucket	GitHub	GitLab	So, what does it mean?
Pull Request	Pull Request	**Merge Request**	In GitLab a request to merge a feature branch into the official master is called a Merge Request.
Snippet	Gist	**Snippet**	Share snippets of code. Can be public, internal or private.
Repository	Repository	**Project**	In GitLab a Project is a container including the Git repository, discussions, attachments, project-specific settings, etc.
Teams	Organizations	**Groups**	In GitLab, you add projects to groups to allow for group-level management. Users can be added to groups and can manage group-wide notifications.

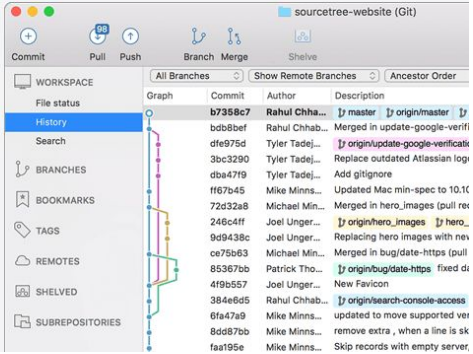
Git software

Sourcetree

Simplicity and power in a beautiful Git GUI

Download for Windows

Also available for Mac OS X



Git GUI

Legendary Git Client for Windows, Mac & Linux

axosoft GitKraken

Legendary Dev Tools

TortoiseGit

Windows Shell Interface to Git

About Download Support Contribute

The Power of Git – in a Windows Shell

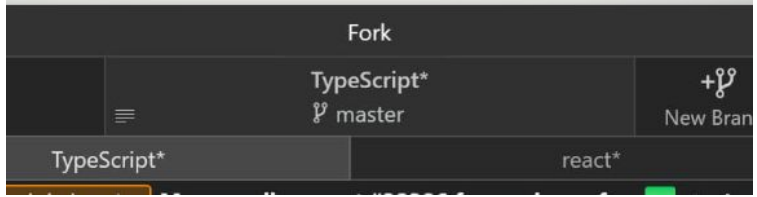
TortoiseGit provides overlay icons showing the file status, a powerful context menu for Git and much more!

Learn more about TortoiseGit.

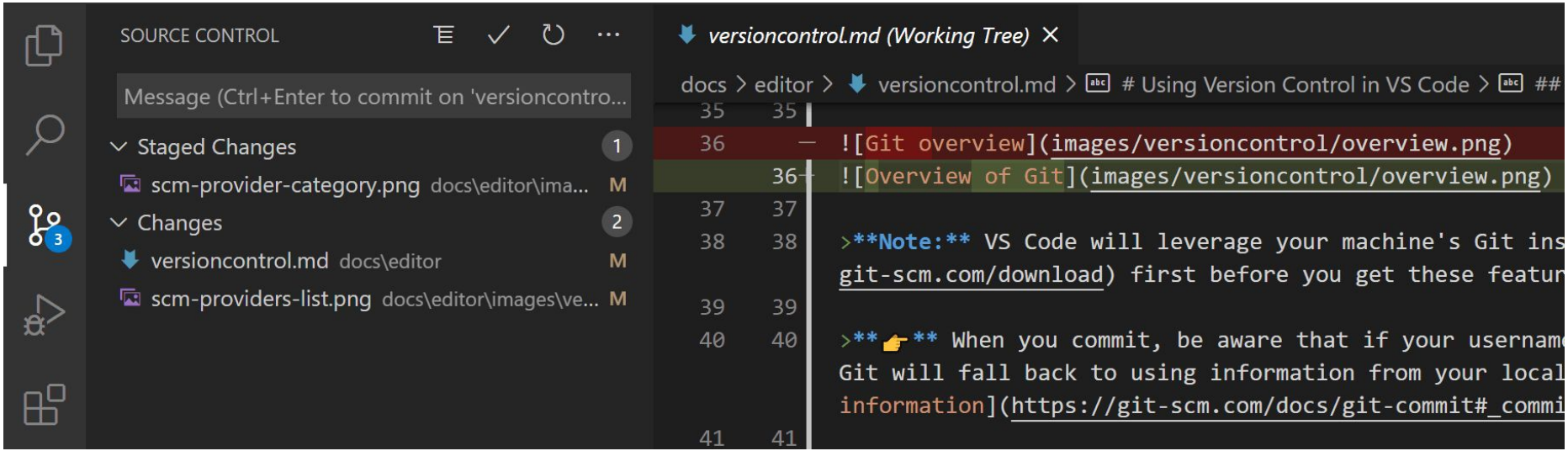
Download

Fork

a fast and friendly git client for Mac and Windows

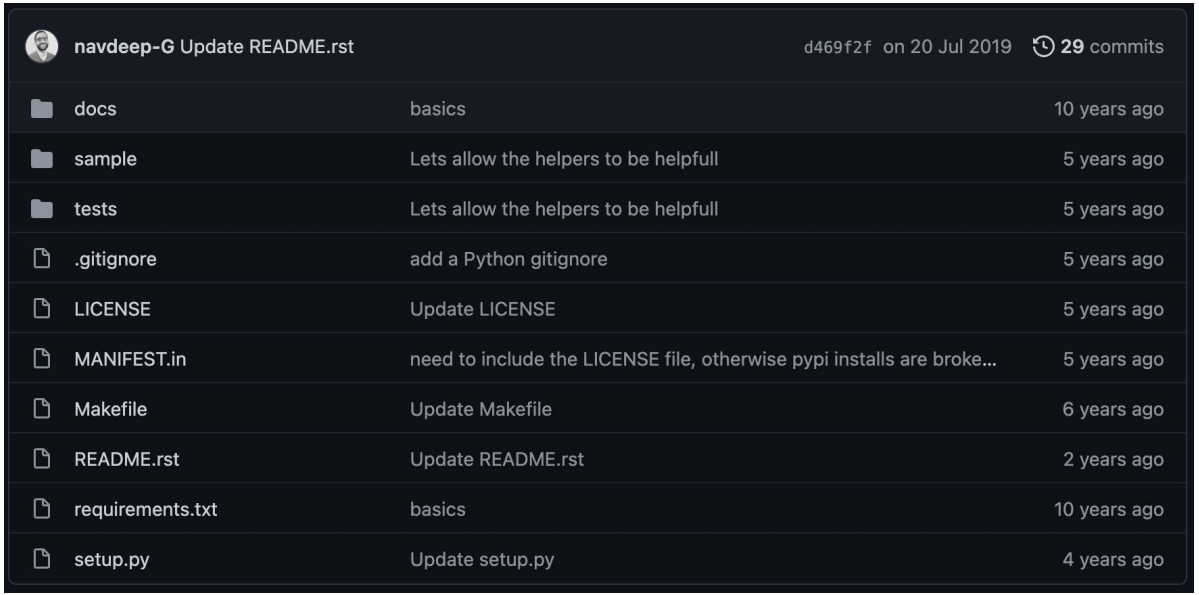


And now...


SOURCE CONTROL
Message (Ctrl+Enter to commit on 'versioncontro...
Staged Changes 1
scm-provider-category.png docs\editor\ima... M
Changes 2
versioncontrol.md docs\editor M
scm-providers-list.png docs\editor\images\ve... M
versioncontrol.md (Working Tree)
docs > editor > versioncontrol.md > # Using Version Control in VS Code > ##
35 35
36 ![Git overview](images/versioncontrol/overview.png)
36 ![Overview of Git](images/versioncontrol/overview.png)
37 37
38 38 >**Note:** VS Code will leverage your machine's Git ins
git-scm.com/download) first before you get these featur
39 39
40 40 >**👉** When you commit, be aware that if your username
Git will fall back to using information from your local
information](https://git-scm.com/docs/git-commit#_commi
41 41

Python projects

Storing Python code



navdeep-G Update README.rst — d469f2f on 20 Jul 2019 — 29 commits

docs	basics	10 years ago
sample	Lets allow the helpers to be helpfull	5 years ago
tests	Lets allow the helpers to be helpfull	5 years ago
.gitignore	add a Python gitignore	5 years ago
LICENSE	Update LICENSE	5 years ago
MANIFEST.in	need to include the LICENSE file, otherwise pypi installs are broke...	5 years ago
Makefile	Update Makefile	6 years ago
README.rst	Update README.rst	2 years ago
requirements.txt	basics	10 years ago
setup.py	Update setup.py	4 years ago

Sample Repository

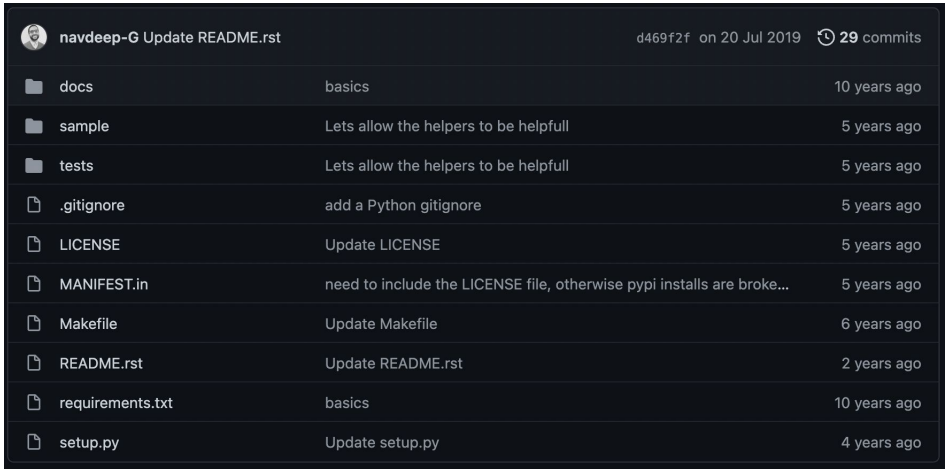
tl;dr: This is what Kenneth Reitz recommended in 2013.

This repository is available on GitHub.

```
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
```

https://docs.python-guide.org/writing/structure/#structure-of-code-is-key

Content of repo



navdeep-G Update README.rst — d469f2f on 20 Jul 2019 — 29 commits

docs	basics	10 years ago
sample	Lets allow the helpers to be helpfull	5 years ago
tests	Lets allow the helpers to be helpfull	5 years ago
.gitignore	add a Python gitignore	5 years ago
LICENSE	Update LICENSE	5 years ago
MANIFEST.in	need to include the LICENSE file, otherwise pypi installs are broke...	5 years ago
Makefile	Update Makefile	6 years ago
README.rst	Update README.rst	2 years ago
requirements.txt	basics	10 years ago
setup.py	Update setup.py	4 years ago

Documentation

Location	`./docs/`
Purpose	Package reference documentation.

The Actual Module

Location	`./sample/` or `./sample.py`
Purpose	The code of interest

Test Suite

For advice on writing your tests, see Testing Your Code.

Location	`./test_sample.py` or `./tests`
Purpose	Package integration and unit tests.

License

Location	`./LICENSE`
Purpose	Lawyering up.

Setup.py

Location	`./setup.py`
Purpose	Package and distribution management.

Requirements File

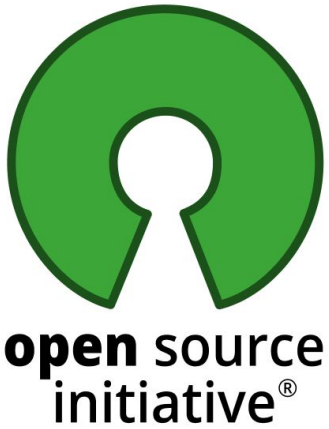
Location	`./requirements.txt`
Purpose	Development dependencies.

Licensing

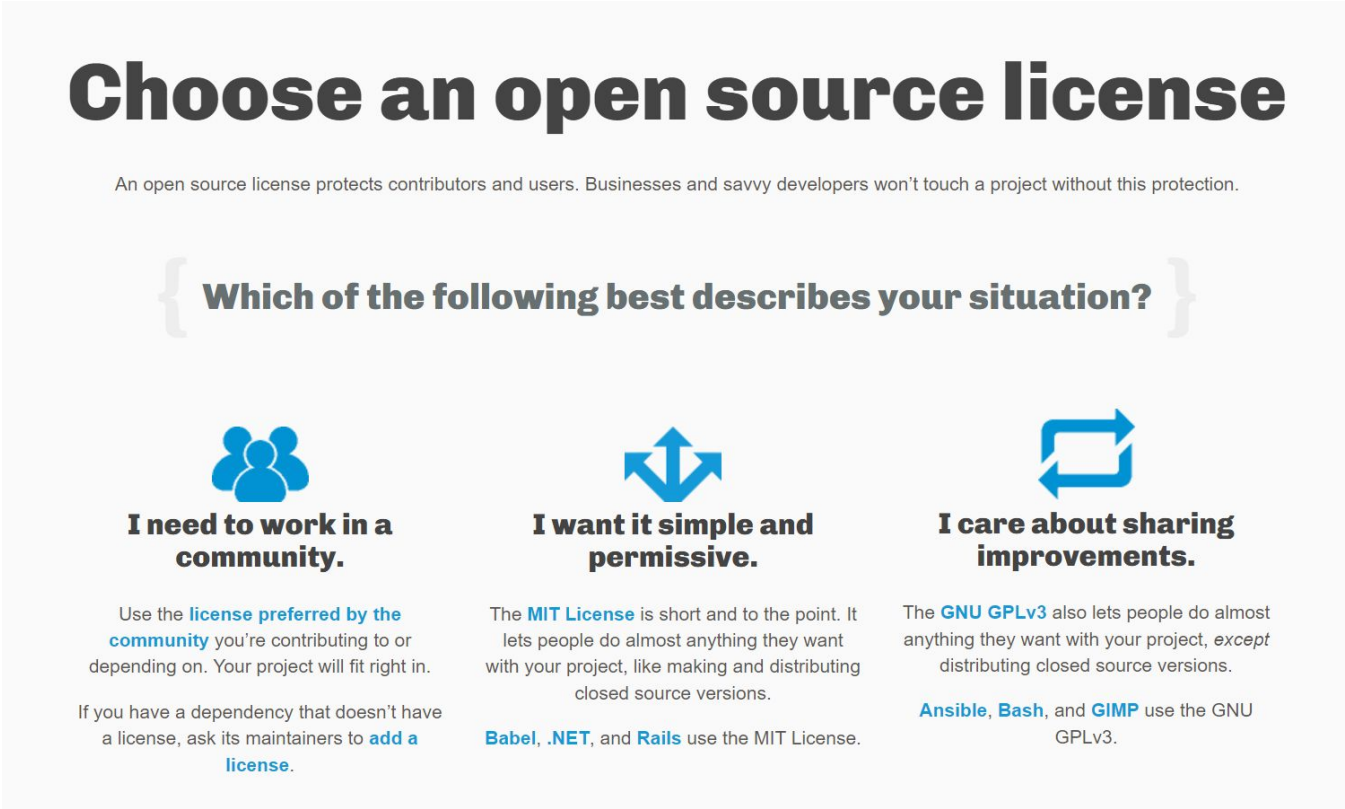
The open source initiative

OSI's Open Source Definition

- free redistribution
- source code availability
- derivatives allowed
- no limitations of who may use it or for what
- no additional license in place
- license must not depend on distribution format, technology, presence of other works



What licenses are around and common?



Choose an open source license

An open source license protects contributors and users. Businesses and savvy developers won't touch a project without this protection.

Which of the following best describes your situation?

I need to work in a community.

Use the **license preferred by the community** you're contributing to or depending on. Your project will fit right in.

If you have a dependency that doesn't have a license, ask its maintainers to **add a license**.

I want it simple and permissive.

The **MIT License** is short and to the point. It lets people do almost anything they want with your project, like making and distributing closed source versions.

Babel, **.NET**, and **Rails** use the MIT License.

I care about sharing improvements.

The **GNU GPLv3** also lets people do almost anything they want with your project, *except* distributing closed source versions.

Ansible, **Bash**, and **GIMP** use the GNU GPLv3.

www.choosealicense.com

Licenses ctd.

1. MIT
   Very lean license
   “Please don’t sue me…” - but very permissive
2. Apache license
   Also very permissive, but with way more words
   ➜ choose if you are afraid of patent trolls
3. GPL (pretty heavy)
   Right choice if you think people use it in an unfair way
   Ensure that derivates are disclosed using the same conditions (also open source)
   Copyleft (instead of copyright)
   Planning in a startup? Then maybe not GPL…

Licenses

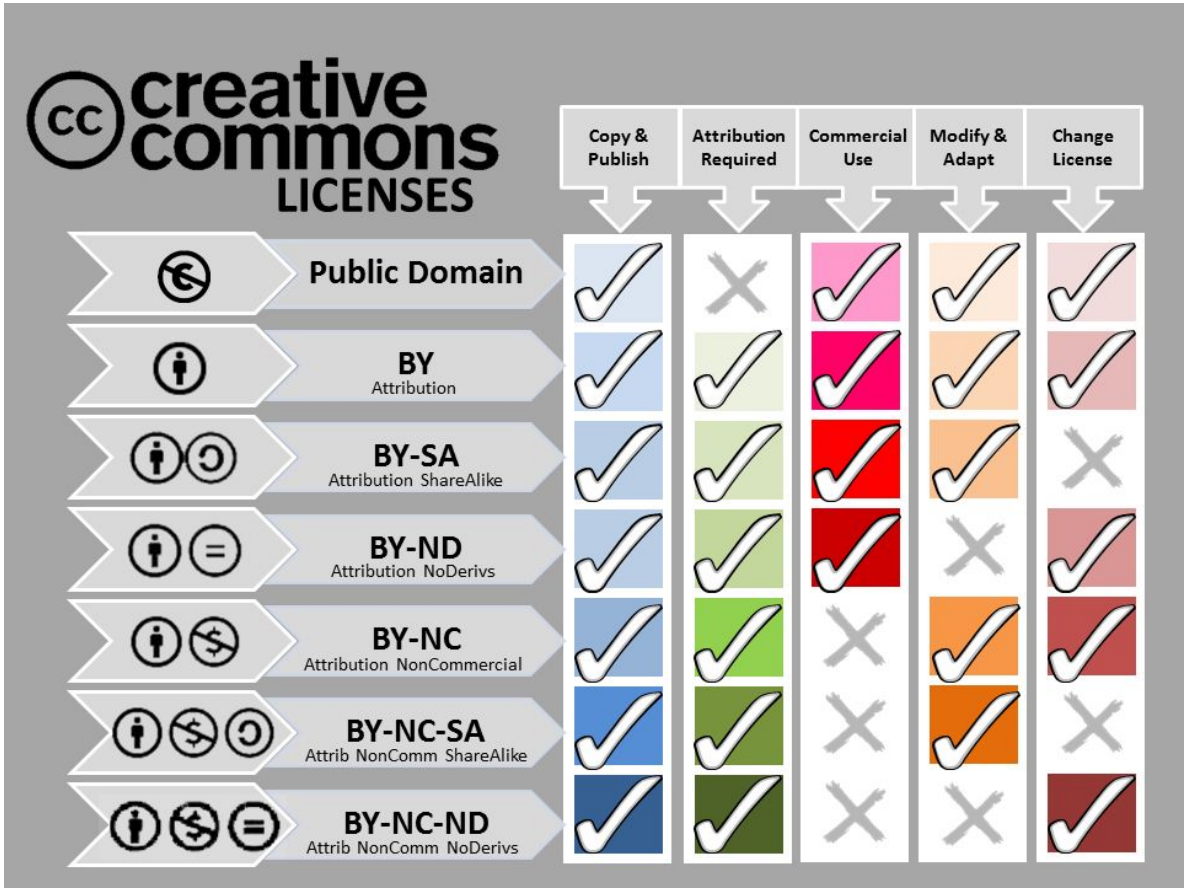
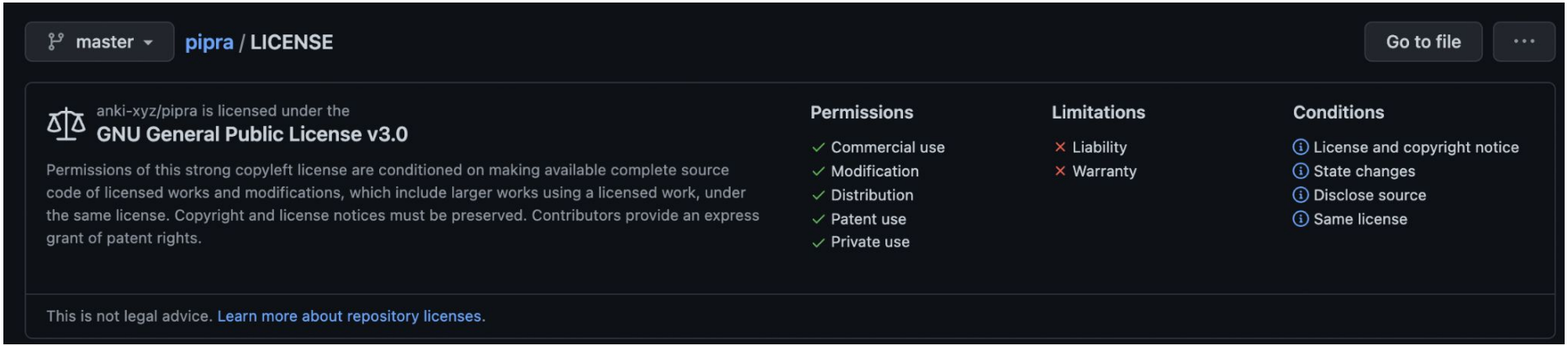


Table 2. Ranking of FOSS licenses' degree of *Openness* based on CC elements.

	Share Alike	No Derives	Noncommercial	Attribution	Ranking of *Openness*
GPL	Yes	No	No	Yes	1
LGPL	Yes	No	No	Yes	1
MPL	Yes	No	No	Yes	1
QPL	No	No	No	Contingent[20]	2
CPL	No	No	No	Contingent	2
Artistic	No	No	No	Contingent[21]	2
Apache v.2.0	No	No	No	Yes	3
zlib	No	No	No	Yes	3
Apache v.1.1	No	No	No	Yes	3
BSD	No	No	No	Yes	3
MIT	No	No	No	Yes	3

https://choosealicense.com/

Licenses on Github



We can’t provide any legal advice here :-)

Homework

Description of the Homework

Welcome to our second homework. With this activity, you will:

- Setup a Git repository.
- Clean-up and improve the code we provide.
- Make your repository pip installable.