

Open Development



Introduction



Open Source

- Publicly accessible source code
- Permission to use, modify, and redistribute
- Driven by transparency and collaboration



Open Development

- Development process is open to the public
- Encourages community participation
- Transparency in decisions, priorities, and changes
- Examples: [Linux](#), [Mozilla Firefox](#), [Kubernetes](#)



Contribution



How to Contribute

- Usually host on platforms like [GitHub](#) or [GitLab](#)
- Fork repositories and create branches
- Submit pull requests and open issues
- Check for a [CONTRIBUTING.md](#) guide



Guidelines and Conduct

- Contribution guidelines (see before)
- Respect the Code of Conduct¹
- Communicate clearly and constructively

1. Examples from [MousePaw Media Standards](#)



Licenses



Why Licenses Matter

- Define how others can use your code
- Protect both contributors and users legally
- Enable sharing with clear rules



Intellectual Property Rights

Two main legal concepts relevant for software:

- **Copyright** = legal ownership
- **Copyleft** = reuse allowed if shared alike
- Balancing freedom with protections



Types of Software Licenses

- **Public domain:** No restrictions on use, modification, or distribution
- **Permissive:** Minimal requirements for reuse (e.g., [MIT](#), [Apache 2.0](#), [BSD-2](#))
- **Copyleft/Share-alike:** Derivatives must use the same open license (e.g., [GPL](#), [LGPL](#))
- **Proprietary:** Use, modification, and redistribution are restricted



Creative Commons (CC) Licenses

grant permission for use of copyrighted creative works

- Applicable to images, videos, presentations, ...
- **CC0**: Waives copyright, placing work in the public domain.
- **CC-BY**: Use allowed with attribution.
- Additional tags: **SA** (share alike), **NC** (noncommercial), **ND** (no derivatives)

check the footer of this slide 😊

Is AI-generated work protected by copyright?

Unclear. Many countries do not put AI-generated work under copyright, but others do (e.g. China).



Documentation



The Value of Documentation

- Helps others understand and use your software
- Onboards new contributors faster
- Can convey intended use and limitations of a tool



What should we document?

intent and usage



Types of Documentation

- **READMEs:** overview, setup, usage
- **Manuals:** deeper technical or user-focused content
- **Docstrings:** inline function/class documentation
- **Comments:** explain *why*, not just *what*



Docstrings

```
1 def sum(a, axis=None, dtype=None, out=None, keepdims=np._NoValue,
2       initial=np._NoValue, where=np._NoValue):
3     """
4     Sum of array elements over a given axis.
5
6     Parameters
7     -----
8     a : array_like
9         Elements to sum.
10    axis : None or int or tuple of ints, optional
11        Axis or axes along which a sum is performed. The default,
12        axis=None, will sum all of the elements of the input array. If
13        axis is negative it counts from the last to the first axis.
14
15        .. versionadded:: 1.7.0
16
17        If axis is a tuple of ints, a sum is performed on all of the axes
18        specified in the tuple instead of a single axis or all the axes as
```

Python docstring conventions ([PEP-257](#))



Manuals

are often build from **docstrings**



User Guide [API reference](#) Building from source Development Release notes Learn [↗](#) More [▼](#)

Section Navigation

NumPy's module structure

Array objects

Universal functions ([ufunc](#))

Routines and objects by topic [^](#)

Constants

Array creation routines

Array manipulation routines

Bit-wise operations

String functionality

Datetime support functions

Data type routines

Mathematical functions with automatic domain

Floating point error handling

[🏠](#) > NumPy reference > ... > Mathematical functions > `numpy.sum`

numpy.sum

`numpy.sum(a, axis=None, dtype=None, out=None, keepdims=<no value>, initial=<no value>, where=<no value>)` [\[source\]](#)

Sum of array elements over a given axis.

Parameters:

`a` : *array_like*

Elements to sum.

`axis` : *None or int or tuple of ints, optional*

Axis or axes along which a sum is performed. The default, `axis=None`, will sum all of the elements of the input array. If `axis` is negative it counts from the last to the first axis.



Commenting Showing Intent (CSI)¹

Do **NOT** state *what* you are doing

```
1 # set box_width to equal the floor of items and 17
2 items_per_box = math.floor(items / 17)
```

DO state *why* you are doing it

```
1 # Divide our items among 17 boxes.
2 # We'll deal with the leftovers later.
3 items_per_box = math.floor(items / 17)
```

1. Examples from [MousePaw Media Standards](#)

Overview

Self-explanatory code	Commenting Showing Intent	Docstrings / Manuals
Actual behavior	Intent & design	Usage & interface
Developers / Maintainers	Developers / Maintainers	End-users / API consumers
What the code does	Why the code exists	How to use it



Data



Data Management

- Preserve research materials for a clearly stated period (typically **10 years**), including:
 - Data and metadata
 - Protocols and other relevant materials
 - **Code and software**
- Ensure access to data is **as open as possible, as closed as necessary**.



(Meta)data and FAIR principles

- **Findable:** unique identifiers, metadata registered in a searchable resource
- **Accessible:** (meta)data retrievable via standardized communication protocol
- **Interoperable:** compatibility with other data (e.g. adhering to [CF conventions](#), use of common data formats like [NetCDF](#), [Zarr](#), [CSV](#))
- **Reusable:** (meta)data description, attributes, data usage license



“FAIR is not fun, but fun is FAIR”

Issues with FAIR data

- data availability not guaranteed
- accessibility only with credentials possible
- **DOIs** don't point to data, but only to landing pages

Beyond FAIR data

- openly accessible
- analysis-ready cloud-optimized data formats ([Abernathey et al., 2021](#))



Good Practices



Clean Code and Version Control

- Write readable, consistent code
- Use meaningful names and modular design
- Commit often with clear messages
- Use branches for features and fixes



Testing and Automation

- Add tests for core functionality
- Use CI/CD pipelines for automated testing
- Build trust through reliability



External libraries

- Use well-established packages when possible
- Usually more robust and efficient
- But you want to understand *what* they are doing 🤔



How to identify a trustworthy source/package?



Supporting the Community

- Use issue and pull request templates
- Document governance and decision-making processes



Keep on learning

stay up to date with coding trends and libraries! 🎓



Summary

- Open development fosters transparency and collaboration
- Contributions thrive with clear guidelines, respectful conduct, and good documentation
- Licenses are essential to protect **and** empower sharing
- It's not just code — it's about **community** and **accessibility**



Further reading

- choosealicense.com
- opensource.guide

