Social coding and open software - What can you do to get credit for your code and to allow reuse

In this lesson we will discuss how and why to share code and data, what kind of licenses are used in what situation, and how software can be cited.

We will try to connect software licenses to FAIR principles, give practical recommendations for starting, contributing, and reusing code and help you navigating and deciding on licenses.

Why software licenses matter

- You **find some great code** or data that you want to reuse for your own publication (good for the original author: you will cite them and maybe other people who cite you will cite them).
- You need to **modify the code** a little bit, or you remix the data a bit.
- When it comes time to publish, you realize there is no **license**.

Now we have a problem:

- You manage to **publish the paper without the software/data** but others cannot build on your software and data and you don't get as many citations as you could.
- Or, you **cannot publish it at all** if the journal requires that papers should come with data and software so that they are reproducible.

This lesson is about how to avoid this situation for you and others.

Prerequisites

No computational prerequisites required for this lesson.

20 min	Social coding
30 min	Software licensing
20 min	Software citation
10 min	Sharing data

Social coding

Objectives

- Get an overview of motivations, benefits, but also risks of sharing and reusing code.
- Learn to identify whether you can use other peoples software.

Discussion/questions/poll: basics of sharing

Instructor note

These questions 1-4 below can be used as a starting point and copied to the collaborative document or form input for an online poll.

Alternatively, they can be discussed in voice or free-form text (discussion box below).

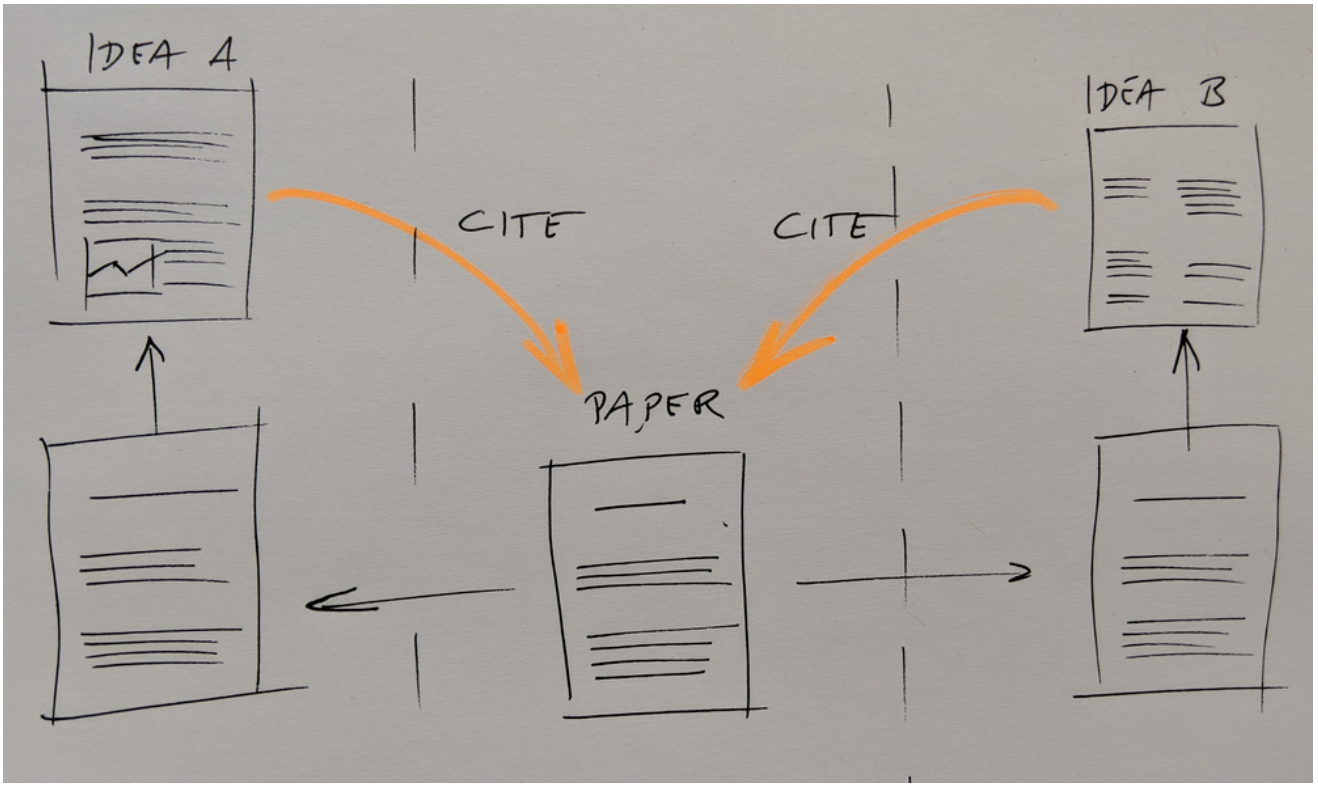
💬 Social-1: Think about if and how you share

- Did you ever share your code? If yes, what motivated you? Come up with **reasons for sharing** your scripts/code/data.
- Also think about **reasons for not sharing**.

```
## Question 1: Why would I want to share my scripts/code/data?

**Choose many**. Vote by adding an `o` character:

- A: Easier to find and reproduce (scientific reproducibility)
  - votes:

- B: More trustworthy: others can verify correctness and find and report bugs
  - votes:

- C: Enables others to build on top of your code
     (derivative work, provided the license allows it)
  - votes:

- D: Others can submit features/improvements
  - votes:

- E: Others can help fixing bugs
  - votes:

- F: Many tools and apps are free for open source, so no financial cost for this
     (GitHub, GitLab, Appveyor, Read the Docs)
  - votes:

- G: Good for your CV: you can show what you have built
  - votes:

- H: Discourages competitors. If others can't build on your work,
     they will make competing work
  - votes:

- I: When publicly shared, usually we time-stamp or set a version,
     so it is easier to refer to a specific version
  - votes:

- J: You can reuse your own code later after change of job or affiliation
  - votes:

- K: It encourages me to code properly from the start
  - votes:


## Question 2: The most concerning thing for me, If I share my software now

**Choose one**. Vote by adding an `o` character:

- A: It will be scooped (stolen) by someone else
  - votes:

- B: It will expose my "ugly code"
  - votes:

- C: Others may find bugs and mistakes. What if the algorithm is wrong?
  - votes:

- D: I will get too many questions, I do not have time for that
  - votes:

- E: Losing control over the direction of the project
  - votes:

- F: Low quality copies will appear
  - votes:
```

```
- G: I won't be able to sell this later. Someone else will make money from it
  - votes:

- H: It is too early, I am just prototyping, I will write version to distribute later
  - votes:

- I: Worried about licensing and legal matters, as they are very complicated
  - votes:


## Question 3: Why is software often treated differently from papers?

Free-form answers:
- ...
- ...
- ...
- ...
- ...
- ...


## Question 4: When you find a repository with code/library you would like to reuse, what are the things you look at to decide whether you use it?

Free-form answers:
- ...
- ...
- ...
- ...
- ...
- ...
```

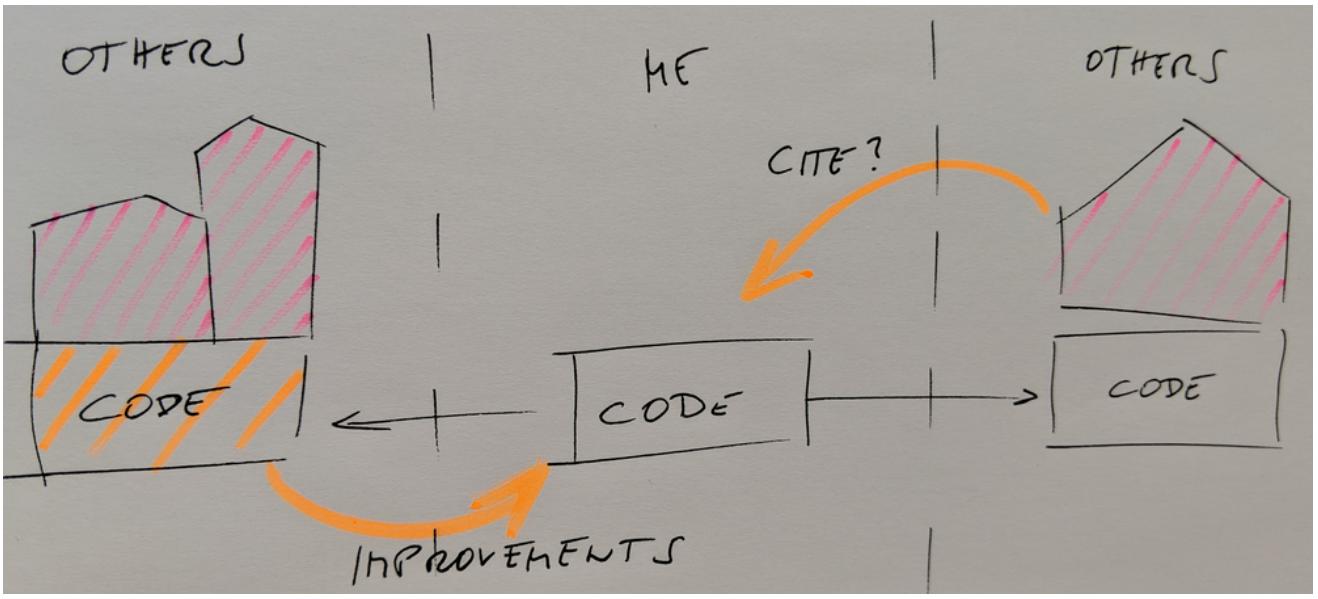
Comparing sharing papers and sharing code



Citation as one form of academic credit to motivate sharing papers.

Sharing papers and academic credit:

- The goal is maximum visibility and maximum reuse.
- The more interesting science is done referencing my paper, the better for me.
- Nobody actively tries to limit the reach of their papers.



Different ways we can benefit from sharing code.

Sharing code:

- "I did all the ground work and they get to do the interesting science?"
- Sharing code and encouraging *derivative work* may boost your academic impact.
- But will your work be visible if it is used two levels deep down?

Journal policies as motivation for sharing

From Science editorial policy:

> "We require that **all computer code used for modeling and/or data analysis** that is not commercially available be deposited in a **publicly accessible repository** upon publication. In rare exceptional cases where security concerns or competing commercial interests pose a conflict, code-sharing arrangements that still facilitate reproduction of the work should be discussed with your Editor no later than the revision stage."

From Nature editorial policy:

> "An inherent principle of publication is that others should be able to replicate and build upon the authors' published claims. A condition of publication in a Nature Research journal is that authors are required to make **materials, data, code, and associated protocols promptly available** to readers without undue qualifications. Any restrictions on the availability of materials or information must be disclosed to the editors at the time of submission. Any restrictions must also be disclosed in the submitted manuscript."

However a study showed that despite these policies, many people still do not share their code 😞. This paper includes samples of charming author responses such as:

"When you approach a PI for the source codes and raw data, you better explain who you are, whom you work for, why you need the data and what you are going to do with it."

Motivation for open source software

Instructor note

We revisit answers to Question 1 (above).

- Enable derivative work
- Do not lock yourself out of own code
- Attract developers who want to be able to show the coding work on their CVs
- Tightly regulated domains require open source
- Open-source software (OSS) can lead to more engagement from industry which may lead to more impact
- If it's not open, it is not likely to become standard

Sharing software is also scary

Instructor note

We revisit answers to Question 2 (above).

- **Fear of being scooped**: A license can avoid it, and you can release when you are ready. Anyway, it is very unlikely that others will understand your code and publish before you without involving you in a collaboration. Sharing is a form of publishing.
- **Exposes possibly "ugly code"**: In practice almost nobody will judge the quality of your code. "Software, once written, is never really finished" (N. Asparouhova).
- **Others may find bugs and mistakes**: Isn't this good? Would you not like to use a code which gives people the chance to locate bugs? If you don't release, people will assume there are bugs anyway.
- **Others may require support and ask too many questions**: This can become a problem: use tools and community and protect your time. You aren't required to support anyone. You can also "archive" a repository to disable most forms of interaction (issues, PRs). Also a note in README on support level helps.
- **Fear of losing control over the direction of the project**: Open source does not mean everybody can change **your version**.
- **"Bad" derivative projects may appear**: It will be clear which is the official version.

Code reusability: What contributes to reusability?

What contributes to you being able to reuse stuff that others make, and others (or you) being able to reuse your stuff? When you find a repository with code you would like to reuse, you may look at the following things to determine its reusability:

Instructor note

This can be now reconnected to Question 4 (above).

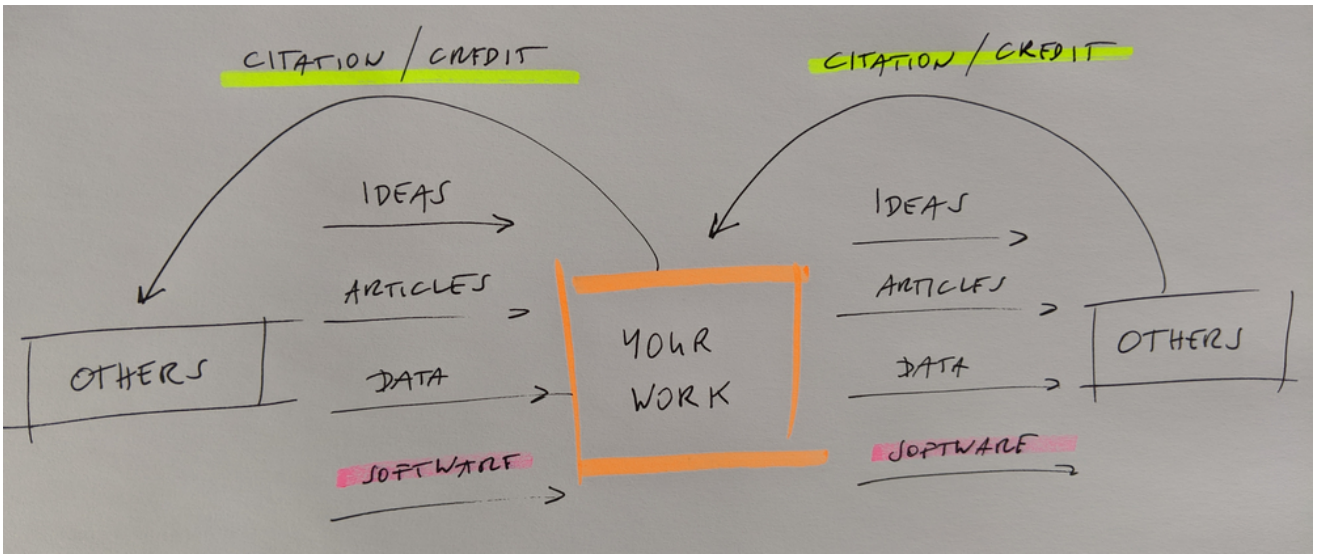
- **Date of last code change**: Is the project abandoned?
- **Release history**: How about stability and backwards-compatibility?
- **Versioning**: Will it be painful to upgrade?
- **Number of open pull requests and issues**: Are they followed-up?
- **Installation instructions**: Will it be difficult to get it running?
- **Example**: Will it be difficult to get started?
- **License**: Am I allowed to use it?
- **Contribution guide**: How to contribute and decision process?
- **Code of conduct**: How to make clear which behaviors are unacceptable and discouraged? How violations of Code of conduct will be handled?
- **Trust and community**: Somebody you trust recommended it?

... most of which you have learned or will learn during this CodeRefinery workshop!

💬 Social-2: Discussion about "You aren't required to support anyone"

- Have you experienced an implicit expectation of support?
- Supporting all requests can lead to overworking and mental health issues.
- Not supporting requests can also induce guilt.
- Most projects are maintained by 1 or 2 persons.
- Most projects cannot retain contributors for a longer time. Interests change. "Casual contributors are like tourists visiting NYC for a weekend" (Nadia Asparouhova, book below).
- If you maintain all projects that you start forever, at some point it may be difficult to start new projects.
- What are your experiences? Do you agree with the above thoughts?
- Book recommendation: Nadia Asparouhova (formerly Nadia Eghbal): "Working in Public: The Making and Maintenance of Open Source Software (Stripe Press)"

How our work connects to the work of others



Whether and what we can share depends on how we obtained the components.

- Our work depends on outputs from others. Research of others depends on our outputs.
- Whether you can share your output depends on how you obtained your input.
- A repository that is private today might become public one day.
- Sometimes “OTHERS” are you yourself in the future in a different group/job.
- Our code often starts by changing other code: **derivative work**.
- **Software licenses** matter. And this is what we will discuss in the next episode.

Software licensing

Objectives

- Knowing about what derivative work is and whether we can share it.
- Get familiar with terminology around licensing.
- Practical advice for software licensing.

Copyright

- **Trademark**: Protects a name/brand from impersonation.
- **Patent**: Protects a novel, non-obvious, technical invention.
- **Copyright**: Protects **creative expression**: software, writing, graphics, photos, certain datasets, this presentation. Practically “forever” (lifetime of author + 70 years).

Copyright controls whether and how we can distribute the original work or the **derivative work**.

Derivative work: Sampling/remixing

[Midjourney, CC-BY-NC 4.0]

[Midjourney, CC-BY-NC 4.0]

- Changing and distributing software is similar to changing and distributing music
- You can do almost anything if you don’t distribute it

Often we don't have the choice:

- We are expected to publish software
- Sharing can be good insurance against being locked out

Exercise: Derivative work

💬 Licensing-1: What constitutes derivative work?

This question 5 below can be used as a starting point and copied to the collaborative document or form input for an online poll:

```
## Question 5: Which of these are derivative works?

**Choose many**. Vote by adding an `o` character:

- A. Download some code from a website and add on to it
  - votes:

- B. Download some code and use one of the functions in your code
  - votes:

- C. Changing code you got from somewhere
  - votes:

- D. Extending code you got from somewhere
  - votes:

- E. Completely rewriting code you got from somewhere
  - votes:

- F. Rewriting code to a different programming language
  - votes:

- G. Linking to libraries (static or dynamic), plug-ins, and drivers
  - votes:

- H. Clean room design (somebody explains you the code but you have never seen it)
  - votes:

- I. You read a paper, understand algorithm, write own code
  - votes:
```

✓ Solution

- Derivative work: A-F
- Not derivative work: G-I
- E and F: This depends on how you do it, see clean room design.

❗ Plagiarism vs. Intellectual Property Rights = Research Ethics vs Law

This insert can be skipped and left as reading exercise

In academic context it is important to consider also *plagiarism* and how it relates to copyright and more broadly Intellectual Property Rights (a clear explanation at this page). Plagiarism is the practice of taking somebody else's ideas or work and claim them as your own: it is the **unacknowledged** use of another person's work. Intellectual Property Rights (IPRs) infringement instead is the **unauthorised** use of another's work.

IPRs can be classified in two main groups (WTO): i) Copyright and rights related to copyright (computer programs are here) and ii) Industrial properties like trademarks, and inventions (which may include specific technical implementations of systems or code) protected by patents.

In research ethics, plagiarism is one of the three definition of research misconduct (along with *fabrication* and *falsification*, see ALLEA, European Code of Conduct for Research Integrity). Plagiarism is not illegal per se, but it can lead to serious consequences like the retraction of published work. One can engage in plagiarism, without necessarily breaking any IPR law (e.g. write a new book by reusing the plot of an old book that is not under copyright anymore). Copyright infringment instead is illegal and it can result in criminal charges (e.g. fines). Copyright however protects the particular expression of an idea or fact (for example, the specific source code of a program, but not the underlying algorithm itself).

There is no pre-defined "number of lines of code", "seconds of a song", or "pixels of an image" that can clearly set the basis for plagiarism or IPR infringement. However in the context of research, it can be possible to use *Quotation Exception* (in EU, ref) and *Fair use* (in USA, ref). Fair use has become controversial recently as it is used as legal basis for training large language models based on scraped internet data (See for example Henderson, P., Li, X., Jurafsky, D., Hashimoto, T., Lemley, M. A., & Liang, P. (2023). Foundation models and fair use. Journal of Machine Learning Research, 24(400), 1-79.)

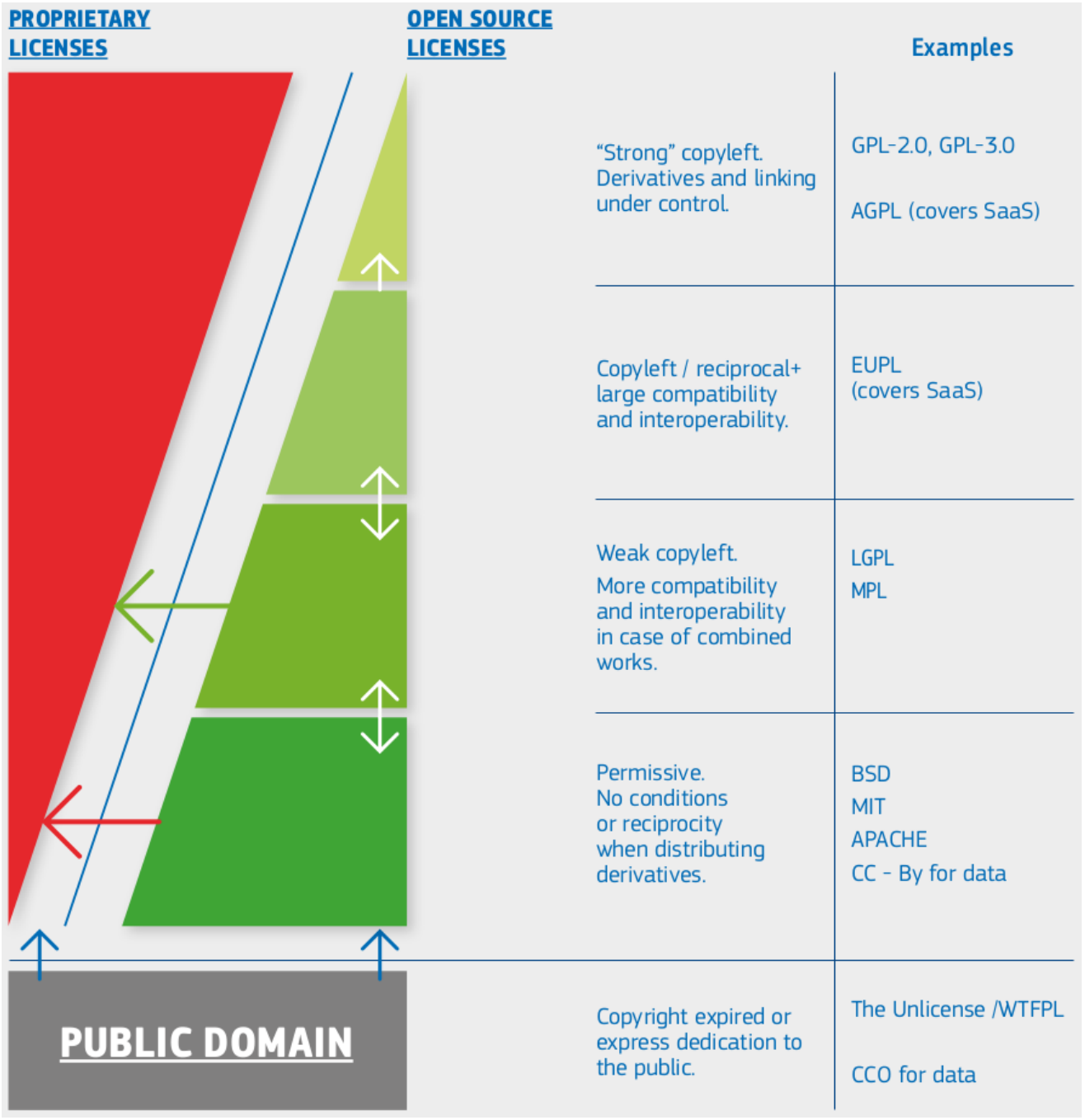
Derivative work and containers

Containers are a bit more tricky when it comes to licenses.

- Distribution of container recipes: it's like distributing source code
- Distribution of container images: it can be considered like distributing a binary compiled software

The latter case is a bit more nuanced and the interested reader should read more about "Mere Aggregation" at GPL-FAQ. Briefly, if the container image just bundles separate programs that talk through normal system interfaces, it is an **aggregate** and each keeps its own license (like a CD-ROM with various packages). If the components are tightly integrated into one program (e.g. a pipeline with various parts that the container can run as a single program), the image may be treated as a **derivative work**, and stricter license obligations (e.g. GPL copyleft) can apply.

Taxonomy of software licenses



European Commission, Directorate-General for Informatics, Schmitz, P., European Union Public Licence (EUPL): guidelines July 2021, Publications Office, 2021, https://data.europa.eu/doi/10.2799/77160

Comments:

- Arrows represent compatibility (A -> B: B can reuse A)
- Proprietary/custom: Derivative work typically not possible (no arrow goes from proprietary to open)
- Permissive: Derivative work does not have to be shared
- Copyleft/reciprocal: Derivative work must be made available under the same license terms
- NC (non-commercial) and ND (non-derivative) exist for data licenses but not really for software licenses

Great resource for comparing software licenses: Joinup Licensing Assistant

- Provides comments on licenses
- Easy to compare licenses (example)
- Joinup Licensing Assistant - Compatibility Checker
- Not biased by some company agenda

If you would like to learn more about licenses, check out our slide deck: "Software licensing and open source explained with cakes".

Exercise: Licensing situations

✍ Licensing-2: Consider some common licensing situations

1. What is the StackOverflow license for code you copy and paste?
2. A journal requests that you release your software during publication. You have copied a portion of the code from another package, which you have forgotten. Can you satisfy the journal's request?
3. You want to fix a bug in a project someone else has released, but there is no license. What risks are there?
4. How would you ask someone to add a license?
5. You incorporate MIT, GPL, and BSD3 licensed code into your project. What possible licenses can you pick for your project?
6. You do the same as above but add in another license that looks strong copyleft. What possible licenses can you use now?
7. Do licenses apply if you don't distribute your code? Why or why not?
8. Which licenses are most/least attractive for companies with proprietary software?

✔ Solution

1. As indicated here, all publicly accessible user contributions are licensed under Creative Commons Attribution-ShareAlike license. See Stackoverflow Terms of service for more detailed information.
2. "Standard" licensing rules apply. So in this case, you would need to remove the portion of code you have copied from another package before being able to release your software.
3. By default you are no authorized to use the content of a repository when there is no license. And derivative work is also not possible by default. Other risks: it may not be clear whether you can use and distribute (publish) the bugfixed code. For the repo owners it may not be clear whether they can use and distributed the bugfixed code. However, the authors may have forgotten to add a license so we suggest you to contact the authors (e.g. make an issue) and ask whether they are willing to add a license.
4. As mentionned in 3., the easiest is to fill an issue and explain the reasons why you would like to use this software (or update it).

5. Combining software with different licenses can be tricky and it is important to understand compatibilities (or lack of compatibilities) of the various licenses. GPL license is the most protective (BSD and MIT are quite permissive) so for the resulting combined software you could use a GPL license. However, re-licensing may not be necessary.
6. Derivative work would need to be shared under this strong copyleft license (e.g. AGPL or GPL), unless the components are only plugins or libraries.
7. If you keep your code for yourself, you may think you do not need a license. However, remember that in most companies/universities, your employer is "owning" your work and when you leave you may not be allowed to "distribute your code to your future self". So the best is always to add a license!
8. The least attractive licenses for companies with proprietary software are licenses where you would need to keep an open license when creating derivative work. For instance GPL and and AGPL. The most attractive licenses are permissive licenses where they can reuse, modify and relicense with no conditions. For instance MIT, BSD and Apache License.

When should I add a license?

Choose a license early in the project, even before you publish it. Later in the project it may become complicated to change it. Agreeing on a software license does not mean that you have to make it open immediately. You can also follow the **"open core" approach**: You don't have to open source all your work. Core can be open and on a public branch. Unpublished code can be on a private repository.

However, we recommend to **work as if the code is public even though it still may be private** (thanks to E. Glerean for this great suggestion): This is to avoid surprises about code in the history with incompatible license years later when you decide to open the project.

How to add a license if your work is derivative work

Your code is derivative work if you have started from an existing code and made changes to it or if you incorporated an existing code into your code.

If your code is derivative work, then **you need to check the license of the original code**. Depending on the license, your choices might be limited. In this case we recommend to use these two resources:

- Joinup Licensing Assistant - Find and compare software licenses
- Joinup Licensing Assistant - Compatibility Checker

If the original code does not have a license, you may not be able to distribute your derivative code. You can try to contact the authors and ask them to clarify the license of their code.

Practical steps for **incorporating something small into your own project** with a license that allows you to do so (as an example incorporating a function or two from another project):

- Create a `LICENSES/` folder in your project and "put the unmodified license text (i.e., the license text template without any copyright notices) in your `LICENSES/` folder" (https://reuse.software/faq/#license-templates). This way if you reuse code from multiple projects, you can keep there multiple license files.
- **Put the code that you incorporate into a separate file or separate files**. This makes it later easier to see what was incorporated, and what was written from scratch. On top of the file(s) which you have incorporated into your project add (and adapt) the following header (more examples):

  ```
  # SPDX-FileCopyrightText: 2023 Jane Doe <jane@example.com>
  #
  # SPDX-License-Identifier: MIT
  ```

  The REUSE initiative was started by the Free Software Foundation Europe to make licensing of software projects easier. It is OK if you prefer to not follow this strict format but the advantage of following it is that the reuse-tool makes it then easy to verify and update license headers if you have many files from different sources.

- If it does not make sense to have several files in your project (e.g. when incorporating something into a notebook), then add a note/comment about the license and where the code came from on top of the function.
- Although it is not dictated by the license but it can still be nice to acknowledge the incorporated functions/code in your README/documentation and to cite their work if you publish a paper about your code.
- Some licenses are more permissive (you can keep your changes private) but some licenses require you to publish the changes (share-alike).

Practical steps for making **changes to an existing project** with a license that allows you to do so:

- If the project is on GitHub or GitLab or similar, first fork the project (copy it into your user space where you can make changes).
- For the BSD and MIT licenses you are not obliged to state your changes but it can still be helpful for others if you do. You can state your changes in the header of the files you have modified. It can be helpful to state bigger-picture changes in the README file of the project.
- Some licenses are more permissive (you can keep your changes private) but some licenses require you to publish the changes (share-alike).

If your work is not derivative work

If you have started “from scratch”, and not used any existing code, or incorporated existing code into your code, then you may consider your code to be not derivative work.

Before you may choose a license, clarify the following points with, for example, your supervisor, collaborators, or principal investigator:

- Does your work contract, grant, or collaboration agreement dictate a specific license?
- Is there an intent to commercialize the code?
- When there is unknown or mixed ownership: If there are multiple persons or organizations as owners of the code, all must agree to the license.

Do not invent your own license. Choose one of the standard licenses, otherwise compatibility is not clear:

- Joinup Licensing Assistant - Find and compare software licenses
- Joinup Licensing Assistant - Compatibility Checker

Practical steps:

- Create a `LICENSES/` folder (example).
- Put the unmodified license text (i.e., the license text template without any copyright notices) in plain text format into the folder (example). Here are the two above licenses in plain text: EUPL and MIT (but the latter contains a copyright notice which we rather want to have on top of files).
- Add copyright and license information to each file following https://reuse.software/tutorial/ which uses a standard format with so-called SPDX identifiers. Example below (example):

  ```
  # SPDX-FileCopyrightText: 2023 Jane Doe <jane@example.com>
  #
  # SPDX-License-Identifier: EUPL-1.2
  ```

  The REUSE initiative was started by the Free Software Foundation Europe to make licensing of software projects easier. It is OK if you prefer to not follow this strict format but the advantage of following it is that the reuse-tool makes it then easy to verify and update license headers if you have many files from different sources.

- For really small projects with one or two files the above may seem excessive and some projects choose to not have copyright information on top of their files and they only have one `LICENSE` file and that is OK for really small projects.

Licensing code produced by generative AI systems

With generative AI tools for coding such as GitHub copilot, Cursor, or even basic chat implementations (ChatGPT, Claude, Grok, ...) the responsibility fully lays on the person who is going to use (and publish) the generated code. You can never blame the autopilot or the company who invented it, only the driver (you!).

There are various risks in using generative AI code (this is not a taxonomy). A few examples:

- Risks for the derivative work: you think your code is doing what you asked, but you did not review it and your results are false
- Risks for the system in use: your generated code has software security issues, e.g. an import is a *typosquat* of an actual library (e.g. "microsoft" is spelled "rnicrosoft" and depending on the font you might totally miss it...)
- Risks related to licenses/IPR: you have generated code that is actually verbatim copy of fully copyrighted code, or code that requires a strict copyleft license. Plagiarism (ethics) also applies.

If we focus on the last one, a recent paper (ref) estimates that around 2% of AI generated code is "strikingly similar to existing open-source implementations". Generative AI tools are typically not able to provide an exact reference of where certain bits of generated code were copied from, so it is the responsibility of the researcher to verify that the produced code is citing and referencing the license of other published pieces of software. Possibly, future AI systems for code generation can be trained on code that share the same set of licenses (e.g. based only on MIT) to mitigate these risks.

Great resources

- Research institution policies to support research software (compiled by the Research Software Alliance)
- Guide from the Aalto University in Finland: "Opening your Software at Aalto University"
- Draft: Research software licensing guide
- Joinup Licensing Assistant - Find and compare software licenses
- Joinup Licensing Assistant - Compatibility Checker
- Social coding lesson material by CodeRefinery
- Citation File Format (CFF)
- License Selector
- GitHub licensing guide
- Choosing an open-source licence
- Understanding Open Source and Free Software Licensing
- Software Licenses in Plain English
- Don's Bibliography of Ethical Source Reading and Resources
- Mikko Välimäki: The Rise of Open Source Licensing
- Lawrence Rosen: Open Source Licensing

- Aalto IPR Cheatsheet
- Contributor License Agreements
- 4OSS recommendations
- 4OSS lesson
- Intellectual Property Rights (IPR), Licensing And Patents
- Dispelling Open Source Confusion: An Introduction to Licenses
- https://choosealicense.com (can send automatic pull request to your GitHub repo)
- https://hintjens.gitbooks.io/social-architecture/content/chapter2.html
- http://rkd.zgib.net/scicomp/open-science/open-science.html
- Nadia Asparouhova (formerly Nadia Eghbal): “Working in Public: The Making and Maintenance of Open Source Software” (Stripe Press)
- Open Source Guides
- The Architecture of Open Source Applications
- Christopher M. Kelty: “Two Bits: The Cultural Significance of Free Software” (Duke University Press, 2008)
- Open Source (Almost) Everything
- 99 ways to ruin an open source project
- Open Source Casebook

Keypoints

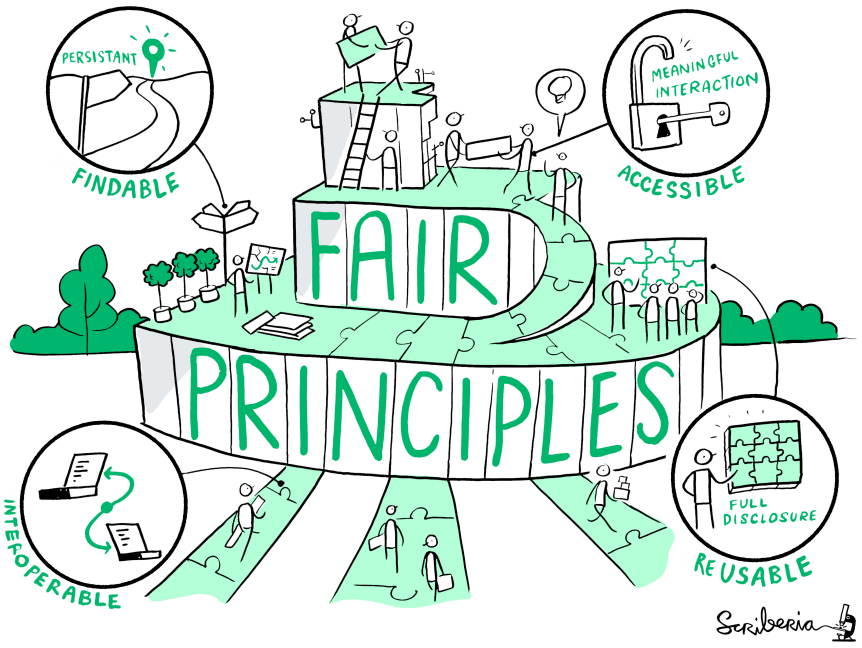
- **You cannot ignore licensing**: default is “no one can make copies or derivative works”.

Software citation

Questions

- Is putting software on GitHub/GitLab/... publishing?
- Where to publish software?
- How can software be cited?
- How can I make my software citeable?

Is putting software on GitHub/GitLab/... publishing?



FAIR principles. (c) Scriberia for The Turing Way, CC-BY.

Is it enough to make the code public for the code to remain **findable and accessible**?

- No. Because nothing prevents me from deleting my GitHub repository or rewriting the Git history and we have no guarantee that GitHub will still be around in 10 years.
- **Make your code citable and persistent**: Get a persistent identifier (PID) such as DOI in addition to sharing the code publicly, by using services like Zenodo or similar services.

How to make your software citable

💬 Discussion (Citation-1): Explain how you currently cite software

- Do you cite software that you use? How?
- If I wanted to cite your code/scripts, what would I need to do?

Checklist for making a release of your software citable:

- ✓ Assigned an appropriate license
- ✓ Described the software using an appropriate metadata format
- ✓ Clear version number
- ✓ Authors credited
- ✓ Procured a persistent identifier
- ✓ Added a recommended citation to the software documentation

This checklist is adapted from: N. P. Chue Hong, A. Allen, A. Gonzalez-Beltran, et al., Software Citation Checklist for Developers (Version 0.9.0). Zenodo. 2019b. (DOI)

Our practical recommendations:

- Add a file called CITATION.cff (example).
- Get a digital object identifier (DOI) for your code on Zenodo (example).
- Step-by-step recipe for how to make your GitHub project citable using Zenodo.
- Make it as easy as possible: clearly say what you want cited.

This is an example of a simple `CITATION.cff` file:

```yaml
cff-version: 1.2.0
message: "If you use this software, please cite it as below."
authors:
  - family-names: Doe
    given-names: Jane
    orcid: https://orcid.org/1234-5678-9101-1121
title: "My Research Software"
version: 2.0.4
doi: 10.5281/zenodo.1234
date-released: 2021-08-11
```

More about `CITATION.cff` files:

- GitHub now supports CITATION.cff files
- Web form to create, edit, and validate CITATION.cff files
- Video: "How to create a CITATION.cff using cffinit"

Papers with focus on scientific software

Where can I publish papers which are primarily focused on my scientific software? Great list/summary is provided in this blog post: "In which journals should I publish my software?" (Neil P. Chue Hong)

How to cite software

Great resources

- A. M. Smith, D. S. Katz, K. E. Niemeyer, and FORCE11 Software Citation Working Group, "Software citation principles," PeerJ Comput. Sci., vol. 2, no. e86, 2016 (DOI)
- D. S. Katz, N. P. Chue Hong, T. Clark, et al., Recognizing the value of software: a software citation guide [version 2; peer review: 2 approved]. F1000Research 2021, 9:1257 (DOI)
- N. P. Chue Hong, A. Allen, A. Gonzalez-Beltran, et al., Software Citation Checklist for Authors (Version 0.9.0). Zenodo. 2019a. (DOI)
- N. P. Chue Hong, A. Allen, A. Gonzalez-Beltran, et al., Software Citation Checklist for Developers (Version 0.9.0). Zenodo. 2019b. (DOI)

Recommended format for software citation is to ensure the following information is provided as part of the reference (from Katz, Chue Hong, Clark, 2021 which also contains software citation examples):

- Creator
- Title
- Publication venue
- Date
- Identifier
- Version
- Type

FAIR for Research Software (FAIR4RS)

The FAIR4RS Principles adapt the original FAIR data principles to improve the **Findability, Accessibility, Interoperability, and Reusability** of research software. They provide guidelines to make research software more open, discoverable, citable, and reusable, which supports reproducibility and open science. The summary of the FAIR4RS below is a compressed version of the principles listed in Barker, Michelle, et al. "Introducing the FAIR Principles for research software." Scientific Data 9.1 (2022): 622.

- **Findable**: Software should have a globally unique and persistent identifier (e.g. clear versioning of releases, DOI from Zenodo when depositing a software release, software metadata in citation file).
- **Accessible**: Software and metadata should be retrievable using open protocols (e.g. downloading from GitHub or Zenodo).
- **Interoperable**: Software should use open formats and standards to work with other tools (e.g. input/output files in CSV, dependencies listed in `requirements.txt` or `environment.yml`, configuration files in yaml).
- **Reusable**: Software should have clear licensing, documentation, and provenance (e.g. a standard license and a `README` with usage instructions, authors listed with ORCIDs).

There are great resources to self-evaluate the *FAIRness* of your research software:

- FAIR Software Checklist which provides a questionnaire and even a badge of how FAIR the software is
- FAIR Software NL highlights with nice visuals the five most important elemnts of FAIR4RS (1. Public accessible repository with version control; 2. License; 3. Software registry; 4. Software citation file; 5. Software quality checklist)

Sharing data

Services for sharing and collaborating on research data

To find a research data repository for your data, you can search on the Registry of Research Data Repositories (re3data) platform and filter by country, content type, discipline, etc.

International:

- Zenodo: A general-purpose open access repository created by OpenAIRE and CERN. Integration with GitHub, allows researchers to upload files up to 50 GB.
- Figshare: Online digital repository where researchers can preserve and share their research outputs (figures, datasets, images and videos). Users can make all of their research outputs available in a citable, shareable and discoverable manner.
- EUDAT: European platform for researchers and practitioners from any research discipline to preserve, find, access, and process data in a trusted environment.
- Dryad: A general-purpose home for a wide diversity of datatypes, governed by a nonprofit membership organization. A curated resource that makes the data underlying scientific publications discoverable, freely reusable, and citable.
- The Open Science Framework: Gives free accounts for collaboration around files and other research artifacts. Each account can have up to 5 GB of files without any problem, and it remains private until you make it public.

Sweden:

- ICOS for climate data
- Bolin center climate / geodata
- NBIS for life science, sequence, omics data

Norway:

- NIRD archive is widely used by some communities
- NSD - Norwegian Center for Research Data, for any kind of data
- Dataverse.no - Dataverse network, based at University of Tromsø but open for other institutions
- ELIXIR Norway for life science, sequence, omics data

Denmark:

- Datavejviser – metadata catalogue aggregating from research organisations and National Archives
- GEUS Dataverse – data from the Geological Survey of Denmark and Greenland
- DTU Data – repository of the Technical University of Denmark
- DeiC Dataverse for University of Copenhagen

Finland:

- IDA, for general data
- FSD for social data
- more information on Finnish resources for open data

Portugal:

- DMPortal - Dataverse network for any research data

Resources for data management

- DataLad
- Data Stewardship Wizard

Licensing of datasets and databases

- The EU has a database directive which restricts data mining on databases.
- Has a somewhat similar effect to copyright, because copyright would not apply to data mining.
- A good license also gives rights to data mine. So not a major concern.

When you can use datasets:

- The license allows
- Your country has exceptions for research
- The data doesn't come from the EU

License text, slides, images, and supporting information under a Creative Commons license, and get a DOI using Zenodo or Figshare or OSF other services.

Licensing and machine learning/ AI

This section is maybe more relevant to **developers of AI models / AI systems** rather than **users of AI models / AI systems**.

Is it data? Is it software? It depends. We need to consider the AI system as a whole, the training data, the production data, the AI output, and how it is put on service. **AI models** are like the engine of the car: they cannot do anything without the rest of the car infrastructure. **AI systems** are the whole car with the AI model and all the software and hardware to actually use it.

Depending on what you are going to share, there might be things to consider beyond the license.

For example **large language models** are often shared with open source software licenses, on **HuggingFace** which is like a GitHub/GitLab for AI models (see for example the OLMO model). Many so-called *open-source* models are actually just *open-weights* models: only the trained neural network weights are shared, while the training data, training code, and full documentation are often kept private. This lack of transparency raises concerns about reproducibility and accountability and this phenomenon is sometimes called **"open washing"** (ref). Models are also shared with a **model card** which is a documentation tool for transparency that provide a comprehensive snapshot of a model's characteristics and ethical considerations (see Ch.8 Glerean 2025).

What about ethics? What about liability?

As AI models (e.g. the deep network weights) and AI systems (the model with all the software and infrastructure to query it) are becoming more available, there can be legal (and ethical!) requirements on the developer of the AI model/system by the EU AI Act. In general researchers do not need to worry, but ethically one should consider that if the research-purpose AI model/system could be used for something harmful, ethically (if not legally) one should consider if such model/system should be implemented at all.

What about the training data inside the model? Large models can memorize and unintentionally reveal parts of their training data. This raises concerns about copyright, trade secrets, and personal data. News publishers and artists are suing AI companies for unauthorized use of their content in training. It is still unclear how traditional data licenses can apply to data that has been transformed into model weights.

More resources

- RAIL initiative: "Responsible AI licenses"
- The Turing Way: Machine Learning Model Licenses
- "Expert Q&A on Artificial Intelligence (AI) Licensing"

Further reading

- The Turing way
- Illustrations from the Turing Way book dashes
- Reproduciblity syllabus
- The reproducible research data analysis platform
- Good talks on open reproducible research can be found here.
- "FAIR is not fair enough"
- "A FAIRer future"

- "Top 10 FAIR Data & Software Things" are brief guides that can be used by the research community to understand how they can make their research (data and software) more FAIR.
- Five recommendations for fair software
- Publishing research software A MIT libraries webpage on why to publish software, where to publish software, and how to make software citable.
- Software Quality Checklist
- MolSSI Best Practice Guides
- Five recommendations for fair software
- Awesome Research Software Registries

Who is the course for?

- Someone writing their own relatively small material
- Someone who wants to incorporate other code/libraries into their own projects
- Group leader who wants to decide how to balance openness and long-term strategy

List of exercises

Full list

This is a list of all exercises and solutions in this lesson, mainly as a reference for helpers and instructors. This list is automatically generated from all of the other pages in the lesson. Any single teaching event will probably cover only a subset of these, depending on their interests.

Instructor guide

Basics

This is basically presented as-is - not much technical presentation is needed.

There are the following main acts:

- First section is about social coding and sharing code and motivation for sharing code with a discussion that licensing alone isn't enough. If you want people to use your work, you need to do more to make your work easily reusable.
- Emphasis of the concept of derivative work and that everything is based on other things. And if your goal is citations, you want as many people using your work as possible, because then they can cite you.
- Overview of license types. There are many online resources, and the biggest question is permissive vs. weak copyleft vs. strong copyleft. The cake analogy for licenses is only linked and can be skipped and does not have to be discussed in detail.
- Then a brief section on code citations. It's good if you manage to start a discussion around this topic.

- The “Sharing data” part is meant to bring awareness about FAIRness and Open Science and to show how DOIs can be obtained for Git repositories.

Hints

Before you begin, it’s worth thinking about some particular good or bad cases you know about from your experience.

- When have you lost access or not been able to use something because licensing wasn’t handled well.
- When did multiple ownership mess up your plans?

This talk used to be called “software licensing”. You could mention that to make it more interesting, we have adopted the current broader name, and our emphasis is not just licensing, but why licensing is needed.