Refreshing previous lecture

Open the Jupyter-notebook of the past lecture and read through it. This will help fixing the learned notions into the long-term memory.

Micro-exercise

After having refreshed the last lecture, switch to the Exercise notebook and complete micro-exercises 1 and 2.

Disclaimer: Today's lecture will be special. In fact, apart from the micro-exercises above it will involve no programming. The lecture will consist in reading the following text and in solving a few exercises along the way.

Licences

An overview on common licences applying to source code, binaries and services, and on their legal background

Definition (software licence): A software licence is a legal document which regulates the *use* and the *distribution* of the licenced software. Some software licences are based on contract law, while most prominent (free and) open-source licences are based on copyright law which is more uniform across different countries than contract law [1]. In almost all countries worldwide, all software (source code or binaries) is automatically protected by copyright, and the copyright is owned by the authors.

Python is not only an open-source programming language (in the sense that the reference implementation of Python is open-source), but it is also the language of choice of uncountable open-source projects. Therefore, it is likely that anybody proficient with Python, will at some point be tempted or compelled to release his code in the "open-source" domain as well.

But what exactly is "open-source software"? Why are there so many different licences? Who are the groups of interests? Who controls it? What is "free software"? These are some of the questions that will be considered in the following.

Observation (UK vs US spelling): In British English, the correct spelling is "licence" (with "c") while in American English the correct spelling is "license" (with "s"). When it comes to the verb, both UK and US English use the form with the "s": to license. In the following, we will use the European form.

[1] https://www.gnu.org/philosophy/no-ip-ethos.en.html

(Free and) open-source software is everywhere

Let's begin with some hard facts which demonstrate how (free and) open-source (FOS) software is permeating our lives, even if we may not realize it during our daily routine. The term "free" will be explained later, but for now it is sufficient to know that it does not relate to the *price* of the software.

- Quoting reference [1]: “Today, **95%** of internet **servers** run on Linux open system. Approximately **85%** of all the **smartphones** run the open-source Android operating system. **Red Hat** [..] has surpassed $2 billion sales a year”. Arguably, the free and open-source Linux “kernel” is one of the most utilized software worldwide.
- The Microsoft cloud service “**Azure**” runs Linux [2] even though Microsoft is a company which develops operating systems for servers.
- Cisco used secretly the free and open-source *GNU Compiler Collection (GCC), GNU Binutils and the GNU C library* and had to pay a 91 M$ fine to the Free Software Foundation for not open-sourcing the derived code [3]. This shows that also wealthy proprietary-software companies find it convenient to utilize free and open-source software.

Over the years, the attitude of companies towards free and open-source software has changed dramatically. For example:

- In the late 90’s open-source was no more just a “nerd” activity but became a threat to big companies. The CEO of Microsoft in 2001, S. Balmer, said for example that “Linux is a cancer” [4]
- In the late 10’s big Companies embraced the free and open-source model, [4]. For example:
  - IBM bought ReadHat for $34 billion in 2018
  - Microsoft acquired GitHub for $7.5 billion in 2018
  - Dice.com incorporated SourceForge for $20 million in 2017
  - Mozilla Foundation revenues raised to $562 million in 2017

[1] https://techtime.news/2016/03/30/semiconductors-4
[2] https://en.wikipedia.org/wiki/Microsoft_Azure
[3] https://www.bizjournals.com/sanjose/stories/2006/08/14/daily75.html and https://en.wikipedia.org/wiki/Free_Software_Foundation,_Inc._v._Cisco_Systems,_Inc.
[4] https://wiki.f-si.org/index.php/Open_Source_Parasitic_Extraction

Choosing a licence

A first question

An important difference between all open-source licences depends from the answer to the following question that, ideally, every developer should consider when choosing a licence. There are cases, however, where the developer is not allowed to choose independently an open-source licence, for example when he is paid by an entity which imposes certain policies. At ETH, for example, there are no clear rules on which open-source licence to use, and the choice is usually made by the principal investigator of a project, or by the supervising professor.

The question is:

Once you publish the source code of your program, do you want to allow somebody to *close* it again? Closing the source code means distributing a program (the binaries) without releasing the source code with it.

- If your answer is no, you should use a “forever-open” licence such as the General Public Licence (GPL). This licence is based on the concept of *copyleft* which will be explained later, and is called *non-permissive* in the sense that it does not allow third parties to close the source again.
- If your answer is yes, you should use a “temporarily-open” licence such as the MIT licence, the BSD licence, or the Apache licence. These licences are called *permissive* because they allow third parties to close the source again.

A second question

Let’s consider the following question which is often neglected on websites such as GitHub (now owned by Microsoft) or by the website choosealicense.com (where GitHub redirects for discussing licences):

Once you publish some source-code do you want to be able to sue its users for infringing a patent that you may own, and which is implemented in the code? Conversely, once you use some source-code, do you accept to be sued by its developer?

At first it may seem absurd that somebody would really consider suing somebody for using his code, but the MIT licence, for example, allows this.

A third question

Once you publish the source-code of your program, is it OK if somebody monetizes on it? This can happen for example by selling copies of the code (with or without source) or by offering services related to the program.

Fortunately, nearly all open-source licences agree on saying that it is OK to monetize on the program. This question therefore is irrelevant in the context of open source. Nonetheless, this question is often (incorrectly) considered important.

Who are the players and what are their interests?

Next, to understand what the different licences try to achieve, let’s consider what are the different groups of interest.
Roughly, we can classify them into two extremes:

On one side we have entities whose primary scope is to:
- do something useful for the general public
- avoid *all* patents
- avoid *all* restrictive standards
- build an open and transparent community
- share as much as possible
- satisfy the personal motivations and interests [1]

To this side belong for example public universities, non-profit organizations, makers or private individuals.

On the other side we have entities whose primary scope is to:

- make profits
- avoid patents of *others*
- avoid standards of *others*
- use gratis the work of everybody (e.g. of universities)
- don’t share everything, or share just the strictly necessary (as obliged by licences or by marketing reasons)

In the second group we typically find large tech companies.

[1]] Lee D. et at., Motivations for Open Source Project Participation and Decisions of Software Developers, Comput. Econ. (41), 31-57 (2012)

The history of open source

The history of open source begins with Richard Matthew Stallman, also known as *rms*. Stallman however refuses, and always refused, to use the term *open source* to make more evident the difference between the freedoms associated with software and the openness of the source code. *Technically*, the free software advocated by rms is also “open source”, but the term *open source* represents historically a different and somewhat orthogonal movement.

Rms graduated in physics at Harvard in 1974 and worked at the MIT Artificial Intelligence Lab from 1971 to 1984 [1]. He founded the Free Software Foundation (fsf.org), launched the GNU project and the Free Software Movement, and he invented the concept of *copyleft* (which will be defined below). He also developed several widely used tools, such as the GNU Compiler Collection (gcc), and GNU emacs [1].

An anecdote of 1980 reveals the atmosphere which led rms to formulate his ideas: in that year, Xerox donated to the MIT Artificial Intelligence Lab a laser printer which was very appreciated, but it had one defect: it was jamming often, but it did not report any error. Rms wanted to modify the software such as to notify the users when the printer jammed, but it was not possible because Xerox shipped only the binaries for the driver, and they refused to share the source code. As a consequence, the printer remained defect causing several nerving situations, despite the fact that in the lab there were many skilled programmers [2, 3].

Another key event which probably shaped the ideas of rms, was the attempt in 1982-1983 by some of his colleagues to close the software which was developed at MIT into a startup.

In 1983 he finally launched the GNU project, in 1984 he quit his paid position at MIT (retaining the affiliation) and founded the Free Software Foundation in 1985.

[1] https://stallman.org/biographies.html#serious
[2] https://www.gnu.org/philosophy/rms-nyu-2001-transcript.txt
[3] https://www.fsf.org/blogs/community/201cthe-printer-story201d-redux-a-testimonial-about-the-injustice-of-proprietary-firmware

The GNU project

The GNU project is a mass collaboration project which was announced by rms in 1983. The term *GNU* is a recursive acronym meaning "GNU's not Unix!"

In the context of the GNU project, rms defined the concept of “free software”. Quoting GNU.org [1]:

<<
“Free software” means software that respects users' freedom and community. Roughly, it means that the users have the freedom to run, copy, distribute, study, change and improve the software. Thus, “free software” is a matter of liberty, not price. To understand the concept, you should think of “free” as in “free speech,” not as in “free beer”. We sometimes call it “libre software,” borrowing the French or Spanish word for “free” as in freedom, to show we do not mean the software is gratis.

The four essential freedoms

A program is free software if the program's users have the four essential freedoms [*]:

- *The freedom to run the program as you wish, for any purpose (freedom 0).*
- *The freedom to study how the program works, and change it so it does your computing as you wish (freedom 1). Access to the source code is a precondition for this.*
- *The freedom to redistribute copies so you can help others (freedom 2).*
- *The freedom to distribute copies of your modified versions to others (freedom 3). By doing this you can give the whole community a chance to benefit from your changes. Access to the source code is a precondition for this.*

[*] The reason they are numbered 0, 1, 2 and 3 is historical. Around 1990 there were three freedoms, numbered 1, 2 and 3. Then we realized that the freedom to run the program needed to be mentioned explicitly. It was clearly more basic than the other three, so it properly should precede them. Rather than renumber the others, we made it freedom 0.
>>

Freedom 1 demonstrates that all free software is technically also open-source software because of freedom 1. However, the term “open source” was coined after the birth of the Free Software Movement and as a reaction to it. Quoting again GNU.org [2]:

<<
That other group of people -- which is called the open source movement -- they only cite the practical benefits. They deny that this is an issue of principle. They deny that people are entitled to the freedom to share with their neighbor and to see what the program's doing and change it if they don't like it. They say, however, that it's a useful thing to let people do that. So they go to companies and say to them, "You know, you might make more money if you let people do this." So, what you can see is that to some extent, they lead people in a similar direction, but for totally different, for fundamentally different, philosophical reasons.

Because on the deepest issue of all, you know, on the ethical question, the two movements disagree. You know, in the free software movement we say, "You're entitled to these freedoms. People shouldn't *stop you from doing these things." In the open source movement, they say, "Yes, they can stop you if you want, but we'll try to convince them to let you to do these things." Well, they have contributed -- they have convinced a certain number of businesses to release substantial pieces of software as free software in our community. So they, the open source movement, has contributed substantially to our community. And so we work together on practical projects. But, philosophically, there's a tremendous disagreement.*

Unfortunately, the open source movement is the one that gets the support of business the most, and so most articles about our work describe it as open source, and a lot of people just innocently think that we're all part of the open source movement. So that's why I'm mentioning this distinction. I want you to be aware that the free software movement, which brought our community into existence and developed the free operating system, is still here -- and that we still stand for this ethical philosophy. I want you to know about this, so that you won't mislead someone else unknowingly.
>>

[1] https://www.gnu.org/philosophy/free-sw.en.html
[2] https://www.gnu.org/philosophy/rms-nyu-2001-transcript.txt

The Open Source Initiative (OSI)

The Open Source Initiative is a California public benefit corporation which was founded in 1998, i.e. several years after the FSF.
Quoting the founding member Michael Tiemann, the OSI chose the use of the term “open source” in contrast to the term “free software” to "dump the moralizing and confrontational attitude that had been associated with free software” and instead promote open source ideas on "pragmatic, business-case grounds” [1].
The goals and the philosophy of the OSI and of the FSF differ significantly, and the two entities have often criticised each others [2].
Guido van Rossum, the author the Python programming language, has been a board member of the OSI in the past.

[1] https://opensource.org/history
[2] https://en.wikipedia.org/wiki/Open_Source_Initiative#Relationship_with_the_free_software_movement

Timeline of major licences

Below are reported the publication dates of the major free-software and open-source software licences. The timeline highlights the premonitory role played by the GNU Project.

Notice: the licences X11, MIT, BSD and Apache represent the “open-source movement” while the GNU General Public Licence (GPL) represent the “free-software movement”.

- 1983: Richard Stallman launches the GNU Project, a mass-collaboration project to create free and open-source software. No licence yet.
- 1988: X11 license, known as MIT X licence or simply MIT licence [1].

- 1988: BSD license (Berkeley Software Distribution).
- 1989: GNU GPL v1 [2].
- 1995: Apache licence 1.0 (Apache foundation/server) [3].
- As reference: GitHub is founded in 2008 only.

[1] https://simple.wikipedia.org/wiki/MIT_License
[2] https://www.gnu.org/licenses/old-licenses/gpl-1.0.en.html
[3] https://www.apache.org/licenses/LICENSE-1.0

Legal background

All software licences are legal documents whose validity is established and enforced by the laws of the different countries. Most software licences are based on the *copyright* law, as opposed for example to *patent* law. Roughly speaking, copyright law protect the *form* of something, while patent law protect an *idea*. For example, if two people write independently a poem about a tree, both poems will almost certainly differ in their form, even if both may describe the same features of the tree, such as its leaves or its stem. Hence the poem is protected by copyright.

Copyright law

Fortunately, even if some details about copyright law may differ in different countries, nearly all countries worldwide share a common denominator thanks to the Copyright Treaty of the United Nation's (UN) World Intellectual Property Organization (WIPO) [1]. This treaty was signed by all UN Member States in 1996 and came in force ~5-10 years later (depending from the countries). The treaty defines also the duration of Copyright protection by referring to the previous "Berne Convention" of 1886 stating that protection lasts at least 50 years, but countries can decide for more.

More specifically, the WIPO Copyright Treaty contains the following articles:

- Article 2 (Scope of Copyright Protection): Copyright protection extends to expressions and not to ideas, procedures, methods of operation or mathematical concepts as such. It does not protect the idea (which is the domain of patents).
- Article 4 (Computer Programs): Computer programs are protected as literary works within the meaning of Article 2 of the Berne Convention [of 1886]. Such protection applies to computer programs, whatever may be the mode or form of their expression.
- Article 7.1 (Right of Rental): Authors of (i) computer programs; (ii) cinematographic works; and (iii) works embodied in phonograms, as determined in the national law of Contracting Parties, **shall enjoy the exclusive right of authorizing commercial rental to the public of the originals or copies of their works**.

Article 2 is important because it defines the exclusive domains of Copyright law and patent law, Article 4 is important because it explicitly extends the protection of Copyright from literary works to computer programs, and Article 7 is important because it says, in other words, that **Copyright law grants to the creator of an original work exclusive rights for use and distribution** (for a limited time).

[1] WIPO Copyright Treaty, http://www.wipo.int/wipolex/en/treaties/text.jsp?file_id=295166

Copyright law is not the same everywhere

As mentioned above, the UN WIPO sets (a very important) common denominator for nearly all countries worldwide. Still, individual countries have the freedom of expanding the protection. Below are some examples showing how Copyright law differs across Europe:

Switzerland:

- The laws are defined in the Urheberrechtsgesetz (URG)
- 70 years protection after author's death. For computer programs: 50 years.

Germany:

- The laws are defined in the Urheberrechtsgesetz (UrhG)
- 70 years on everything

EU:

- The EU attempts to harmonize the law of the Member States through a number of directives

How copyright works

Copyright law has been developed primarily to *prohibit* people from using, modifying and copying a work. It is thanks to rms that copyright law has been *hacked* into something which actually guarantees perpetual rights of doing the exact opposite, namely of using, modifying and copying a work - a mechanism that rms called *copyleft* to highlight its opposite goals (more on it later).

Let's take as an example the movie industry (see figure below). Warner Bros, for example creates a movie. As the copyright holder of the movie, *Warner Bros is granted exclusive rights for use and distribution*. Warner Bros then says: "Don't copy the movie! All rights are reserved". This allows to distribute a movie copy, like a DVD, to Julian, but Julian is forbidden to make a copy for Sarah.`

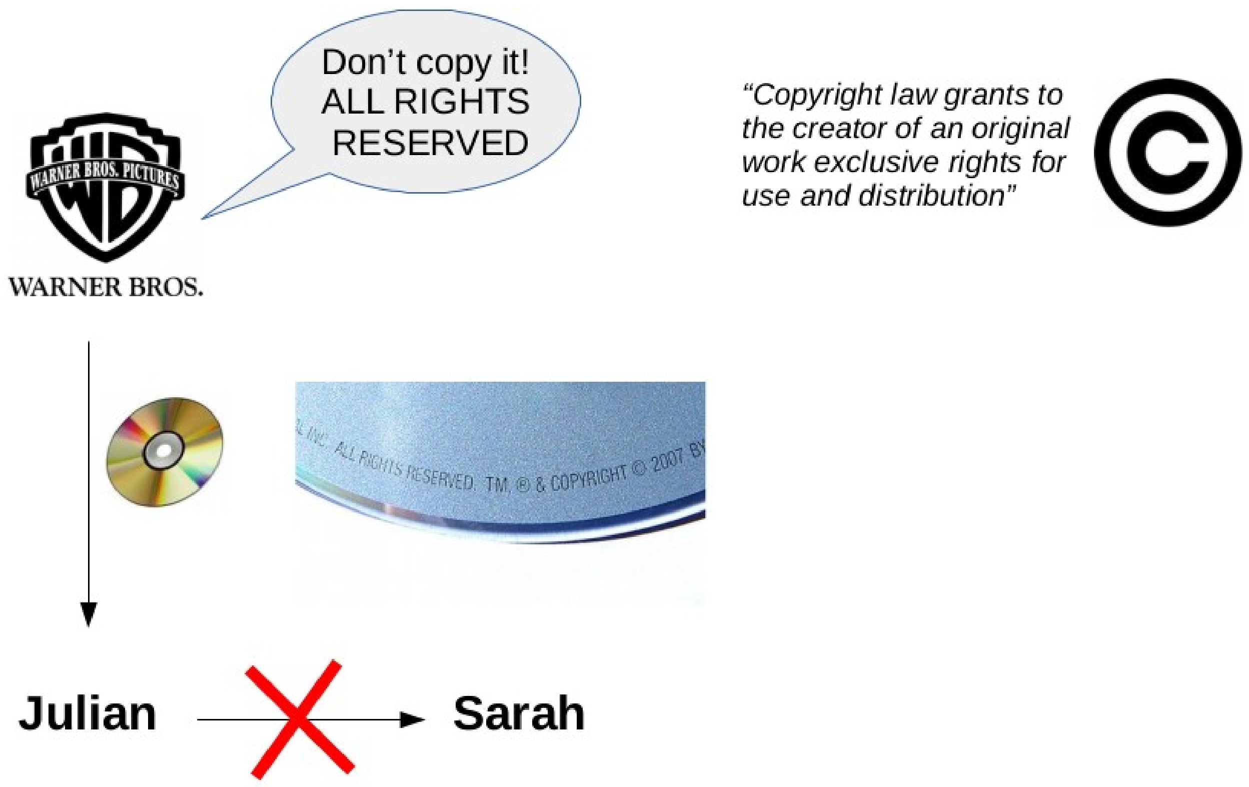

Don't copy it! ALL RIGHTS RESERVED
WARNER BROS. PICTURES
WARNER BROS.
"Copyright law grants to the creator of an original work exclusive rights for use and distribution"
ALL RIGHTS RESERVED. TM ® & COPYRIGHT © 2007 BY
Julian
Sarah

Figure 1: How copyright works for protecting illegal copies of movies.

CopyLEFT uses copyright to achieve the opposite

Copyleft utilizes copyright laws to grant perpetual freedom to all users of a program. In contrast to a complete surrender of all copyright rights (as in the case of a *public domain attribution*), copyleft retains certain rights in order to impose certain obligations on those using the code and therefore guarantee that the code remains forever-open.

Let's look how the most famous copyleft licence, the General Public Licence (GPL) works in practice. Under copyright law, the original copyright holder has the *exclusive rights of use and distribution.* He therefore grants to everybody the right to use and distribute the program provided that he obeys to all the requirements listed in the GPL text. These conditions are literally *comply-or-die* conditions in the sense that if any of these conditions are not met, then the use and the copying of the code becomes automatically illegal. It is like if the original copyright holder says: "Don't use nor copy it! Unless you give the source and the same licence again" (see image below).

The conditions contained in the GPL text are twofold:

- the first set of conditions grants rights to the users, and in particular grant them all of the four fundamental software freedoms
- the second set of conditions impose obligations and say that whenever the code is copied (either in its original or in a modified form), then it must be copied with the same licence again and again. This condition is similar to the mathematical proof by induction where if a condition is valid for N, then it is valid also for N+1.

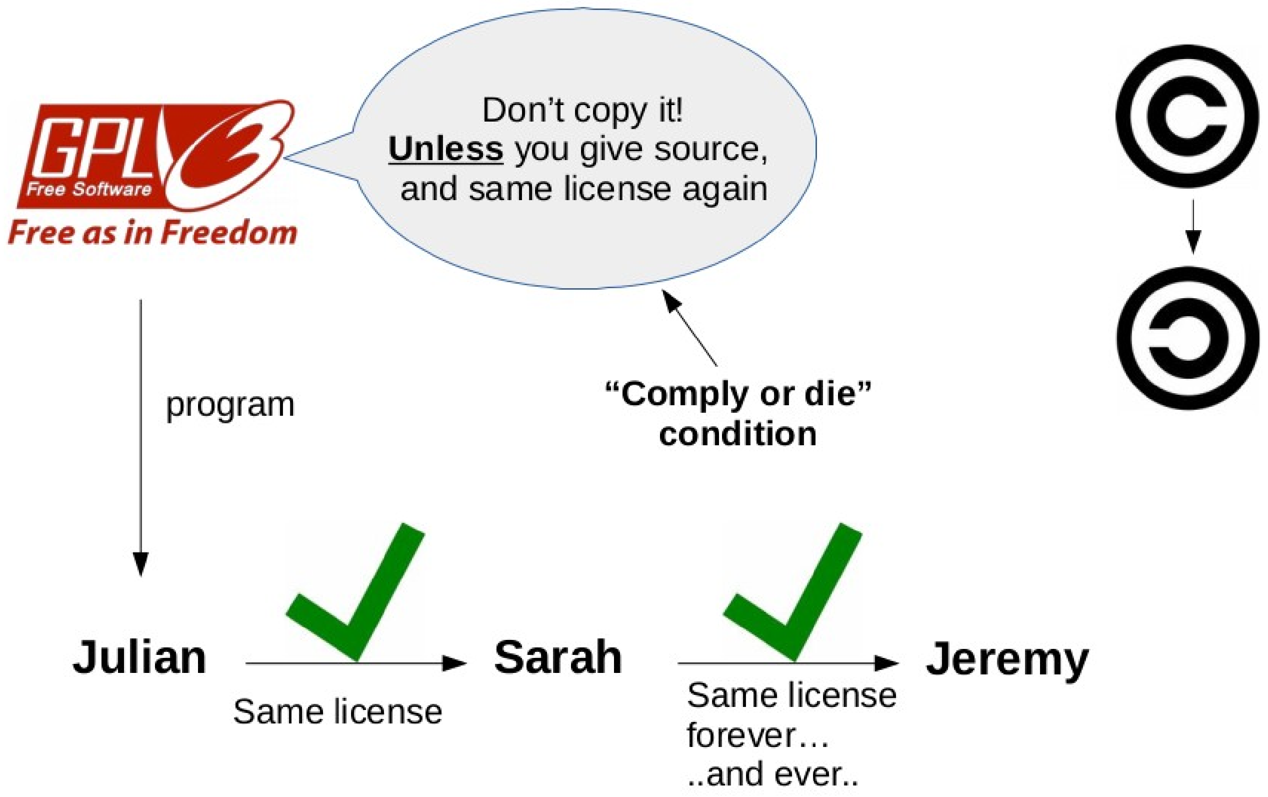


Figure 2: Illustration of how copyleft works: the original copyright holder allows copying provided that the same rights are passed down the line therefore guaranteeing perpetual permission to copy and obligation to release the source code as well.

The symbol of copyleft is a backward C in a circle, and it is therefore the mirror-inverted image of the copyright symbol. This inversion can represent symbolically the opposite goals of copyleft when compared to copyright.

Even though the copyleft symbol has no formal legal meaning, the symbol has been accepted by the Unicode Technical Committee and is available as "U+1F12F".

Example: John Doe's "coffee licence"

Here we want to illustrate the "comply-or-die" condition which is used in copyleft licences to enforce obligations. Since the original author of a work holds the exclusive rights for use and distribution of his work, he can in principle mandate any sort of condition. A crazy condition is imposed by the "coffee licence" below which basically says "I allow you to use my code *provided that* you give me coffee":

Copyright (C) 2018 John Doe
This software is coffee software. You can use it and modify it provided that you
offer one cup of coffee to the author every day you use the software.
You may convey a work based on the software provided that you license the
entire work, as a whole, under this license to anyone who comes into possession
of a copy.

```
class powermeter_XYZ:
    def __init__(self, host='xyz.ee.ethz.ch', averaging_time_us = 1000):
        self.host = host
        self.averaging_time_us=averaging_time_us

    def __del__(self):
        self.socket.close()
        print('Connection at ', self.host, 'closed and object deleted.')
...
```

The case of servers: the Affero General Public Licence (AGPL)

The GPL licence requires people to convey the source code (i.e. the "preferred form of the work for making modifications to it") whenever a program is *conveyed* (in its original or in a modified form).

But what if some "free software" is executed on a remote server without conveying any program to other users, but by just delivering some services?

If the author of the code wishes that the source code is made available also in such a scenario, the obligation of conveying the source code can be *triggered* at a different point than when conveying a program. This is achieved by the Affero GPL (or GPL). In the AGPL is written in fact:

<< *[The AGPL] requires the operator of a network server to provide the source code of the modified version running there to the users of that server.* >>

In other words, when you visit the website of a server running AGPL software, you should find a link to the source code or a hint on how to find it.

Example: dudle is a free-software alternative to the popular doodle service and it is available on several public instances such as https://dudle.hu-berlin.de/ .

Micro-exercise

At this point, switch to the Exercise notebook and complete micro-exercise **3**.

Permissive licences

The term "permissive licence" has been defined by the Open Source Initiative (OSI). *A permissive licence is by definition any non-copyleft licence.*

The term "permissive" means that third parties, like companies, are "permitted" to close the source code of a (formerly open source) program. Permissive licences are well received by many companies, such as Google. In fact, some companies even ban non-permissive licences. Google for example writes:

<<*WARNING: Code licensed under the GNU Affero General Public License (AGPL) MUST NOT be used at Google*>> [1]

Permissive licences allow to transfer public-funded code (for example code developed at universities) to companies which can then modify, sell or use the program without releasing the source code any more. A famous example of permissive-licenced code is the BSD kernel which was first developed also by an enthusiastic and unpaid programmers, and which became then part of the closed-source iOS operating system of Apple.

Typical permissive licences are the BSD licence, the MIT licence and the Apache licence.

There are also companies which explicitly avoid permissive licences. Nextcloud, for example, utilizes only AGPL to maintain independence from investors and for increasing transparency with the customers as explained by a talk of the founder Frank Karlitschek [2].

[1] https://opensource.google/docs/using/agpl-policy/

[2] Why I forked my own project and my own company ownCloud to Nextcloud, Frank Karlitschek, https://conf.tube/videos/watch/05857289-4f99-4109-bd71-f5e0a779a544

Comparison of different licences

In the table below are reported some of the most common open-source licences. Roughly speaking, they are sorted from less free (left) to more free (right). The last column is unrelated and represents proprietary software as a comparison. The first column represents possibly the worse of all possible scenarios in terms of right protections for the original author which is releasing some code without any form of copyright notice (left-most column).

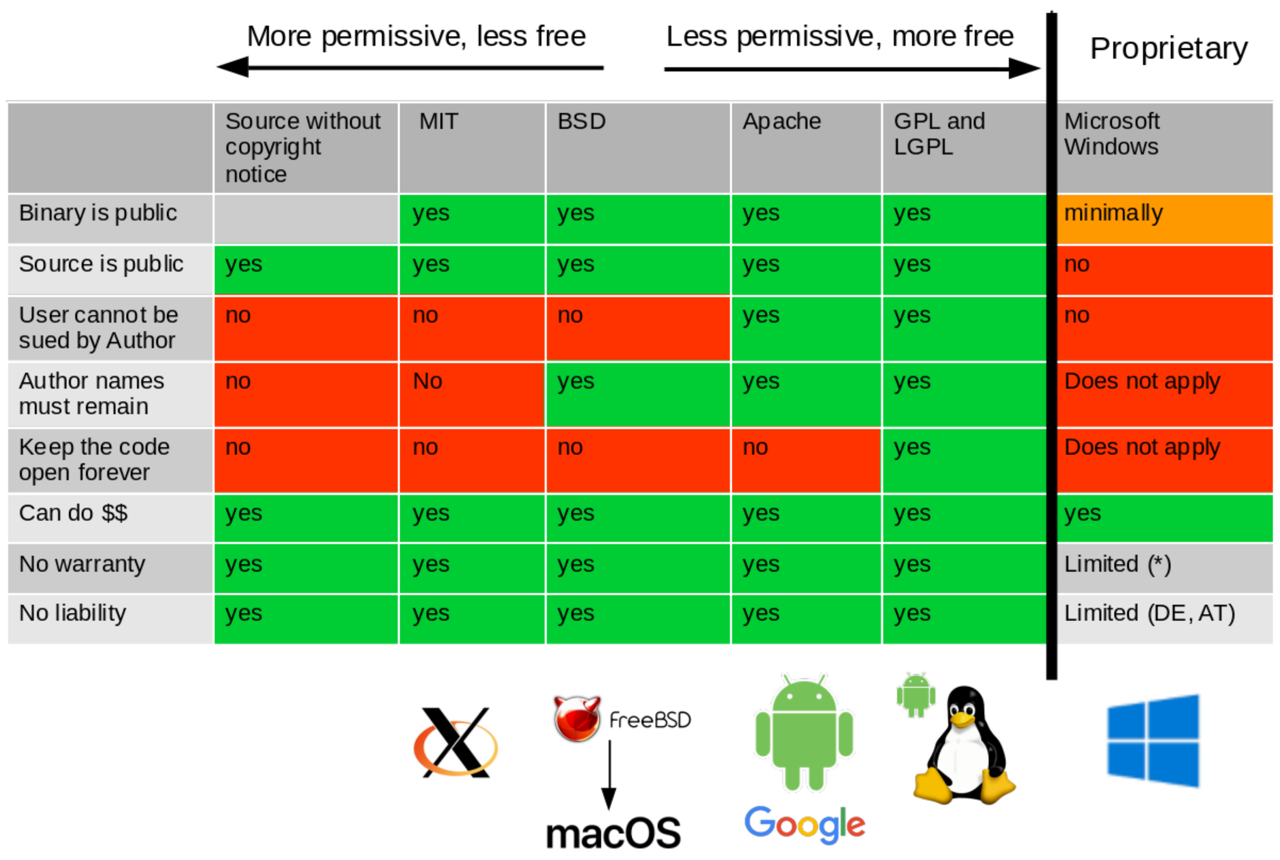
From the table it is apparent that only the Apache and the GPL licence contain a “patent clause” which automatically forbids the original authors to sue the users of the published code for patent infringement. When Google chose the Apache licence for developing Android, for example, it automatically protected itself from being sued by third parties who may have introduced source code protected by a patent.

Many programmers care whether their name must be preserved in the source-code or not. In the MIT licence for example the name can simply be erased from the source-code. With the BSD and with the Apache licence instead the name of the authors must remain in the source-code. However, the source-code can be removed all together (by closing the source code) which is nearly equivalent. Only the GPL licence guarantees that the author names will remain forever attached to a program.

With the MIT, BSD and Apache licences the source-code can be closed any time. It is therefore more appropriate to call such licences “temporarily-open” as suggested in [1].

As mentioned at the beginning of the lecture, most open-source licences agree about the right of monetizing with the use of the program. There are no differences either when it comes about warranty (for example: the code does not work as advertised, hence the author has to fix it) or liability (for example: I had an accident because the code had a bug, hence the author has to pay for it).

[1] https://wiki.f-si.org/index.php/White_paper_for_the_EC,_January_2020


More permissive, less free
Less permissive, more free
Proprietary
FreeBSD
macOS
Google

	Source without copyright notice	MIT	BSD	Apache	GPL and LGPL	Microsoft Windows
Binary is public		yes	yes	yes	yes	minimally
Source is public	yes	yes	yes	yes	yes	no
User cannot be sued by Author	no	no	no	yes	yes	no
Author names must remain	no	No	yes	yes	yes	Does not apply
Keep the code open forever	no	no	no	no	yes	Does not apply
Can do $$	yes	yes	yes	yes	yes	yes
No warranty	yes	yes	yes	yes	yes	Limited (*)
No liability	yes	yes	yes	yes	yes	Limited (DE, AT)

(*) Microsoft warrants that properly licensed software will perform substantially as described in any Microsoft materials that accompany the software.

Figure 3: Comparison of different licences from less free (left) to more free (right). Under each columns are the icons of some programs adopting each licence. The last column is unrelated and represent proprietary software as a comparison.

Controversial licence advertisement on popular platforms

Since the birth of the Internet, the web has become more and more *centralized*, with few giant services dominating most of the activities online. Think for example about Gmail for emails, Facebook for social media, or GitHub for “open-source” development.

As centralized platforms gain in popularity, they also gain power of influencing the public opinion which can sometimes be dangerous. Quoting rms, for example, “*GitHub's encouragement of sloppy licensing, no licensing, or licensing under only a single version of the GPL, has done terrible harm to our community.*” [1]

Micro-exercise

At this point, switch to the Exercise notebook and complete micro-exercise **4** to verify in practice what rms meant with the sentence above.

[1] https://lists.gnu.org/archive/html/gnu-system-discuss/2019-11/msg00277.html

GPL vs. Lesser GPL (LGPL)

If you receive a program licenced under GPL, then you have the assurance that the *entirety* of its source-code is available to you. This includes in particular all libraries, meaning both the *statically* linked libraries (which are those whose binaries will be inside the final executable) and the *dynamically* linked libraries (which are those which do not end up inside the final executable and could be shared for example among different programs inside an operating system; they usually terminate with *.so* or *.dll*). Only in this way the complete freedoms of the users can be guaranteed.

In other words, if you want to release a program under GPL, you cannot include any proprietary library. But what if there is no open-source library available which does what you need?

In this case you can use the Lesser GPL which explicitly allows you to include closed-source *dynamically* linked libraries. In this case the freedoms are not fully guaranteed to the user, as the term “Lesser” represents. Notice that the *statically* linked libraries need to remain open, and there is no GPL-like licence allowing to close them as well.

How-to: a practical guide to add a licence to your code

This section will follow closely the recommendation from the GNU project [1]:

In short:

- Each file should include the copyright notice. Use the full word “Copyright” and not abbreviations such as (c). Also add the year of the last change to the file and the name of the copyright holder.
- Each file in the project should contain a licence notice (copy it from the creator of your license). This can be a short text, namely not the full licence text.
- Somewhere in the distribution of your program, you should include the full text of the licence used. Call this file “COPYING” when using GPL.

[1] https://www.gnu.org/licenses/gpl-howto.en.html

Exercises

At this point switch to the Exercise notebook and complete Exercise **1** and **2**, then upload your solutions as usual.