



Python modules and packages

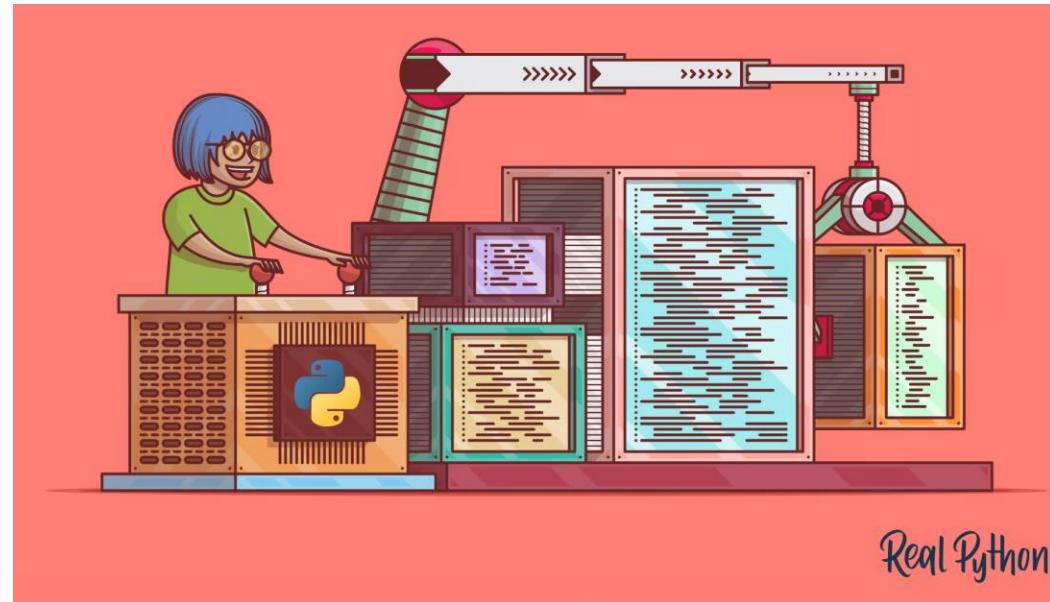
Structuring large(r) projects

Fulvio Corno

Giuseppe Averta

Carlo Masone

Francesca Pistilli



<https://realpython.com/python-modules-packages/>

Goal

- Split a large program in multiple files
- Make re-usable library of classes
- Import and use additional libraries

Modules

- Modules are collections of classes, functions, variables, and declarations
Il modulo viene importato con `import!`
- The `import` statement runs the module source and makes the definitions available
- The defined names are created in a separated *namespace* to avoid confusion

Un modulo è un qualunque file Python che contiene al suo interno delle funzioni, variabili, ecc. che può essere importato in un altro programma. Importando un modulo in un programma, posso utilizzare all'interno del programma i metodi, variabili, funzioni ecc definiti all'interno del modulo.

Esempi di moduli che posso importare nel mio programma, senza doverli scaricare!

Standard library modules

- random, math, csv, operator, os, dataclasses, datetime, ... 200 more
- <https://docs.python.org/3/library/index.html>

N.B.: Se ho due file, non posso da un file accedere alle classi dell'altro; dovrò prima importarlo nel file principale!

User-defined modules

- My python sources
- Files and directories in my project
- Files and directories in the Python search path

↳ Posso anche importare moduli importati da me stesso!

Downloaded modules

- From <https://pypi.org/> - over 500k projects
- Install using pip
Con pip posso scaricare ≠ moduli fatti da qualcun'altro!

Creating a Python module

- Just create a `.py` file
 - In the `same directory` of your main file
 - Should contain `declarations, only`
- The `name of the file` is the name of the module
 - The argument of `import`
- All names defined `at the top-level` become visible properties of the module
 - Constants
 - Functions
 - Classes
 - Variables (bad idea)

Per creare un modulo in Python basterà creare un file .py NON dotato di main all'interno della quale potrò mettere eventuali classi, funzioni, variabili globali ecc. Questo sarà un modulo.
Se voglio utilizzare questo modulo con le sue classi, funzioni, ecc. all'interno del main (che si dovrà trovare nella stessa cartella del modulo), basterà importare nel file main.py il modulo, scrivendo `import ... (nome che ho dato al modulo)`

```
voto.py
```

```
MAX_VOTO = 30

class Voto:
    def __init__(self, esame, cfu, punteggio, lode, data):
        ...
    def __str__(self):
        ...
    def __repr__(self):
        ...

class Libretto:
    def __init__(self):
        ...
    def append(self, voto):
        ...
    def media(self):
        ...

def voto_casuale():
    ...
```

```
main.py
```

```
import voto
```

The import statement

- `import module_name`
 - Imports the definitions from `module_name`. They will be accessible as `module_name.definition`


Prima di usare un metodo di un modulo importato, dovrò scrivere il nome del modulo importato, con la notazione puntata, e poi il nome del metodo!
 - Example: `import math; use math.sin(math.pi)`
- `import module_name as alt_name` ↗
Posso anche dare un nome ≠ al modulo importato nel mio file!
 - Imports the definitions from `module_name`. They will be accessible as `alt_name.definition`
 - Example: `import cmath as c ; use c.sqrt(-1)`

The from...import statement

- `from module_name import name(s)` ↗
– Import one or more name(s) from `module_name`, and make them available in the current namespace
– Example: `from math import pi, sin, cos ; use sin(pi)`
 - `from module_name import name as alt_name`
– Import one name from `module_name`, and make it available in the current namespace as `alt_name`
– Example: `from cmath import sqrt as csqrt ; use csqrt(-1)`
 - `from module_name import *` ↗
– Import all available names from `module_name`, and make them available in the current namespace
– Except names starting with '`_`' (underscore)... they are ignored
– Somewhat dangerous... may have conflicts with local names or other module's names
- Posso importare **selettivamente** quello che mi serve del modulo, e non tutto il modulo!
In questo caso non dovrò scrivere `Nomemodulo.Nome.metodo` x usare un metodo del modulo importato,
ma potrò evitare di mettere davanti il nome del modulo!
- Questo equivale a importare l'intero modulo => Anche in questo caso non dovo metterci il nome del modulo
davanti, però non va fatto perché si rischiano **conflicti**!

Querying available names: `dir()`

- The `dir()` function shows the list of names defined in a module
 - `dir()`: names defined (at the top level) of the `current` file
 - `dir(module_name)`: names defined in the `imported` module
- Several *dunder* methods, plus user-defined names

• La funzione `dir()` mostra quali sono i nomi definiti all'interno di un programma.

↗ Per nomi si intende:
• moduli
• variabili
• classi
• ecc.

```
import voti
print(dir(voti))
# ['Libretto', 'MAX_VOTO', 'Voto', 'random', 'voto_casuale', ...]

print(dir())
# ['voti', ...]
```

```
from voti import Voto, Libretto
print(dir(voti))
# NameError: name 'voti' is not defined.

print(dir())
# ['Libretto', 'Voto', ...]
```

È utile ad esempio x scoprire
queli sono i metodi definiti
all'interno del modulo/programma.

A module should not contain executable statements

- The `import` statement **runs the module file** to create the definitions of the various names
- If there are any instructions **outside** the defined classes/functions, **they will be executed, too...**

Quando importo un modulo in un file, tutto il codice del modulo viene automaticamente eseguito

NON VA BENE!

Quando nei moduli che importo dovrò avere solo classi / funzioni, senza avere delle istruzioni eseguibili!

Le istruzioni eseguibili devono trovarsi solamente nel main!

- If `voti.py` contains

```
v = voto_casuale()  
print(repr(v))
```
- Then, the `import voti` statement in `main.py` will cause
 - Defining a new top-level name (`v`)
 - Calling `voto_casuale()`
 - Printing the random vote
- All this should **not** happen!

Solving the problem

- It is useful to have some code *inside the module*
 - Usually, **test code** to verify that the module works correctly
 - Sometimes, a **whole program** (with its top-level code) may be **used as a module** for a larger problem
- We **want** to **allow** in-module code, but we **don't want** it to **run**, when imported

In un modulo è quindi meglio avere solamente classi/funzioni e non istruzioni eseguibili così che quando importo quel modulo in un altro file non venga eseguito il codice al suo interno. Se volessi però comunque avere delle righe di codice eseguibili ma che NON vengano eseguite quando importo il modulo, LO POSSO FARE.
Per poterlo fare devo creare all'interno del modulo una funzione `_main()` dove aggiungo tutte le istruzioni eseguibili. Dopo di che aggiungo l'istruzione `_main()`

Questo perchè all'interno di ogni modulo viene definita la variabile `__name__`. Questa variabile assume il valore del nome del modulo stesso se mi trovo in un altro file che sta importando il modulo. Invece se mi trovo nel modulo stesso, assume il nome `__main__`.
Esempio: se ho un modulo `voti.py` che importo in `main.py`, la variabile `__name__` del modulo `voti`, all'interno del file `main` che lo sta importato assumerà il valore `'voti'`. Mentre se lo printo all'interno del modulo stesso `voti`, assumerà il valore `__main__`.
Aggiungendo queste istruzioni, quando importo il modulo dato che `__name__ != "__main__"`, allora le istruzioni eseguibili del modulo non verranno eseguite. Mentre se voglio runnare il programma all'interno del modulo stesso, dato che `__name__ == "__main__"`, allora verrà chiamata la funzione `_main` con tutte le istruzioni eseguibili del modulo!

- The solution is to check if the file is the top-level one, or an imported one: `__name__`

```
if __name__ == "__main__":
    v = voto_casuale()
    print(repr(v))
```

- Or, better:

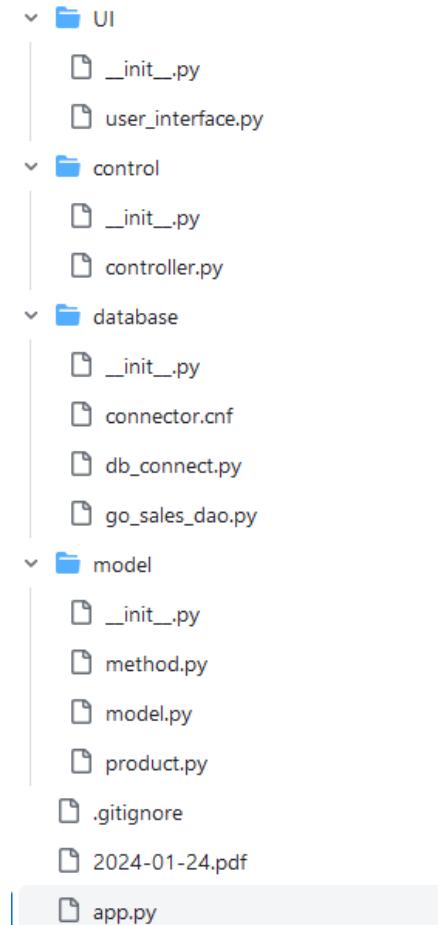
```
def _main():
    v = voto_casuale()
    print(repr(v))

if __name__ == "__main__":
    _main()
```

Packages

- When an application grows, it is no longer viable to have all the Python file in a **single** directory
- We can split groups of files in separate directories, called **packages**
 - Each **directory** is a **package**
 - The **files** of the directory are **modules**
 - They can be imported with the syntax **package_name.module_name**

I moduli sono file nella stessa cartella del file in cui li sto importando. Se l'applicazione è grossa, allora può risultare utile organizzare i moduli in package, che sono insieme di moduli.
Se voglio importare un modulo che fa parte di un package (per intenderci, i package sono delle cartelle contenenti file Python), dovrò usare la sintassi `package_name.module_name`, dove il nome del package è il nome della cartella, e il nome del modulo è il nome del file Python.



(and sub-packages)



Importing from packages

- The traditional syntax still applies
 - import pkg.mod
 - from pgk.mod import name
 - from pgk.mod import name as alt_name
 - Additionally, you may import modules from a package
 - from pkg import mod
 - from pkg import mod as alt_mod_name
- Nuova sintassi × gestire l'import di moduli utilizzando i package!

The `__init__.py` file

- Traditionally, the directory containing a package will also contain a special file
 - `__init__.py`
 - It was mandatory until Python 3.3, now it's optional
- Can contain initialization statements, that are run when importing any module from the package

Tutti i package contengono un file `__init__.py`, che può essere anche vuoto, e serve a inizializzare il package.
Fino a tempo fa era obbligatorio, adesso è FACOLTATIVO.

→ Serviva a Python a riconoscere
il package!

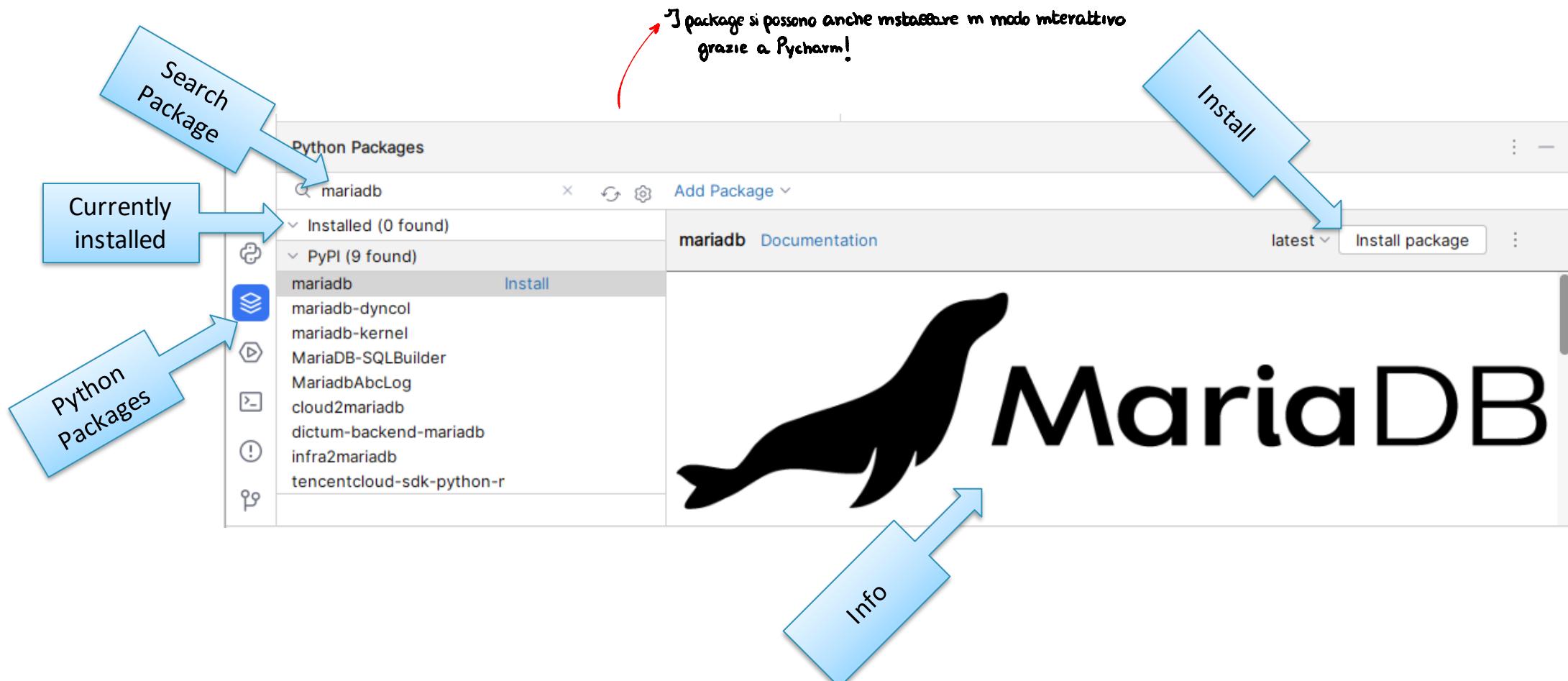
Working with external modules

- To access a module from pypi.org, we must first install the module in our local Python interpreter
- Packages can be installed using the **pip** program
 - Search a project on <https://pypi.org/>
 - Install with **pip install project_name**
 - pip install flet
 - pip install mariadb
- Only installed packages can be imported

Posso anche installare moduli esterni tramite **pip!**

Dopo aver installato moduli/librerie esterne, potrò **importarle** nel mio programma!

Installing packages with PyCharm



Virtual Environments

- Different projects may require different packages
- ! Your local Python library will contain all sorts of packages, that are used by some project
- ! When shipping a project, it's not clear which packages are needed to run it

N.B.: Quando creo un nuovo progetto, viene creato automaticamente anche un virtual environment!

Un Virtual Environment è un meccanismo in Python che ci permette di isolare i package esterni di cui abbiamo bisogno, definendoli per progetto.

- Python has a mechanism for separating the packages needed by each project
- ! Virtual environments
- It's a local “copy” of the Python interpreter, alongside with the packages needed for that project
- Stored in the .venv directory

Il virtual env viene copiato nella cartella venv.

Virtual Environments in PyCharm

The image shows two screenshots of the PyCharm interface illustrating how to set up a virtual environment.

For existing project: This screenshot shows the "Python Interpreter" settings dialog. It lists interpreters "Python 3.12" and "Python 3.11". A button "Add New Interpreter..." is highlighted. A dropdown menu is open, showing options: "Add Local Interpreter...", "On SSH...", "On Vagrant...", "On WSL...", "On Docker...", and "On Docker Compose...". A blue arrow points from this dropdown down to the "Create" button in the "New Project" dialog below.

For new project: This screenshot shows the "New Project" dialog. It has "Name: pythonProject" and "Location: D:\teaching\TdP-2024". Under "Interpreter type", "Project venv" is selected. Under "Python version", "Python 3.12.0 C:\Users\Fulvio\AppData\Local\Programs\Python\Python312\python.exe" is chosen. A "Create" button is at the bottom right.

Environment: Existing (radio button) New

Location: D:\teaching\TdP-2024\libretto_voti\.venv

Base interpreter: Python 3.12 C:\Users\Fulvio\AppData\Local\Programs\Python\Python312\python.exe

Inherit global site-packages

Where does Python find packages?

- The import statement searches packages
 - In the current project directories
 - In the current virtual environment's library
 - In a set of directories defined by the Python installation

```
import sys  
print(sys.path)
```

```
[ 'C:\\\\Users\\\\Fulvio\\\\AppData\\\\Local\\\\Programs\\\\PyCharm  
Professional\\\\plugins\\\\python\\\\helpers\\\\pydev',  
'C:\\\\Users\\\\Fulvio\\\\AppData\\\\Local\\\\Programs\\\\PyCharm  
Professional\\\\plugins\\\\python\\\\helpers\\\\third_party\\\\thri  
fty',  
'C:\\\\Users\\\\Fulvio\\\\AppData\\\\Local\\\\Programs\\\\PyCharm  
Professional\\\\plugins\\\\python\\\\helpers\\\\pydev',  
'C:\\\\Users\\\\Fulvio\\\\AppData\\\\Local\\\\Programs\\\\PyCharm  
Professional\\\\plugins\\\\python\\\\helpers\\\\pycharm_display',  
'C:\\\\Users\\\\Fulvio\\\\AppData\\\\Local\\\\Programs\\\\Python\\\\Pyt  
hon312\\\\python312.zip',  
'C:\\\\Users\\\\Fulvio\\\\AppData\\\\Local\\\\Programs\\\\Python\\\\Pyt  
hon312\\\\DLLs',  
'C:\\\\Users\\\\Fulvio\\\\AppData\\\\Local\\\\Programs\\\\Python\\\\Pyt  
hon312\\\\Lib',  
'C:\\\\Users\\\\Fulvio\\\\AppData\\\\Local\\\\Programs\\\\Python\\\\Pyt  
hon312', 'D:\\\\teaching\\\\TdP-2024\\\\libretto_voti\\\\.venv',  
'D:\\\\teaching\\\\TdP-2024\\\\libretto_voti\\\\.venv\\\\Lib\\\\site-  
packages',  
'C:\\\\Users\\\\Fulvio\\\\AppData\\\\Local\\\\Programs\\\\PyCharm  
Professional\\\\plugins\\\\python\\\\helpers\\\\pycharm_matplotlib  
_backend', 'D:\\\\teaching\\\\TdP-2024\\\\libretto_voti']
```

Dove Python cerca i progetti che voglio importare!

requirements.txt

Il file requirements.txt è utile per far conoscere alle persone esterne i package che è necessario installare per poter utilizzare correttamente il nostro progetto.
Questo perchè quando creo un virtual environment io installo i moduli e le librerie esterne all'interno del virtual environment del progetto, e se condivido il progetto ad esempio su GitHub, una persona terza che utilizza il progetto su un'altra macchina e un altro virtual environment non avrà installato automaticamente tutti i moduli/librerie, e quindi devo scrivere tutti i progetti installati all'interno di un file requirements.txt così che anche persone terze possano installarle prima di runnare il programma.

- A project may require several external packages
 - Installed with pip
 - Stored in the virtual environment
- How can we declare the information about the required packages?
 - So that other people may install them in their system
 - So that we can control which version numbers are installed
- Add a file **requirements.txt** to your project
 - Contains one line per package
 - May optionally specify the version number
 - PyCharm helps us synchronizing the file with the import statements

```
requirements.txt
1 pip==22.3.1
2 wheel==0.38.4
3 Pillow==9.5.0
4 setuptools==65.5.1
5 packaging==23.1
6 numpy==1.26.0

Don't specify version
Strong equality (==x.y.z)
Greater or equal (>=x.y.z)
Compatible version (~=x.y.z)
```



License

- These slides are distributed under a Creative Commons license “**Attribution-NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0)**”
- **You are free to:**
 - **Share** — copy and redistribute the material in any medium or format
 - **Adapt** — remix, transform, and build upon the material
 - The licensor cannot revoke these freedoms as long as you follow the license terms.
- **Under the following terms:**
 - **Attribution** — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
 - **NonCommercial** — You may not use the material for commercial purposes.
 - **ShareAlike** — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.
 - **No additional restrictions** — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.
- <https://creativecommons.org/licenses/by-nc-sa/4.0/>