

# Python modules and packages

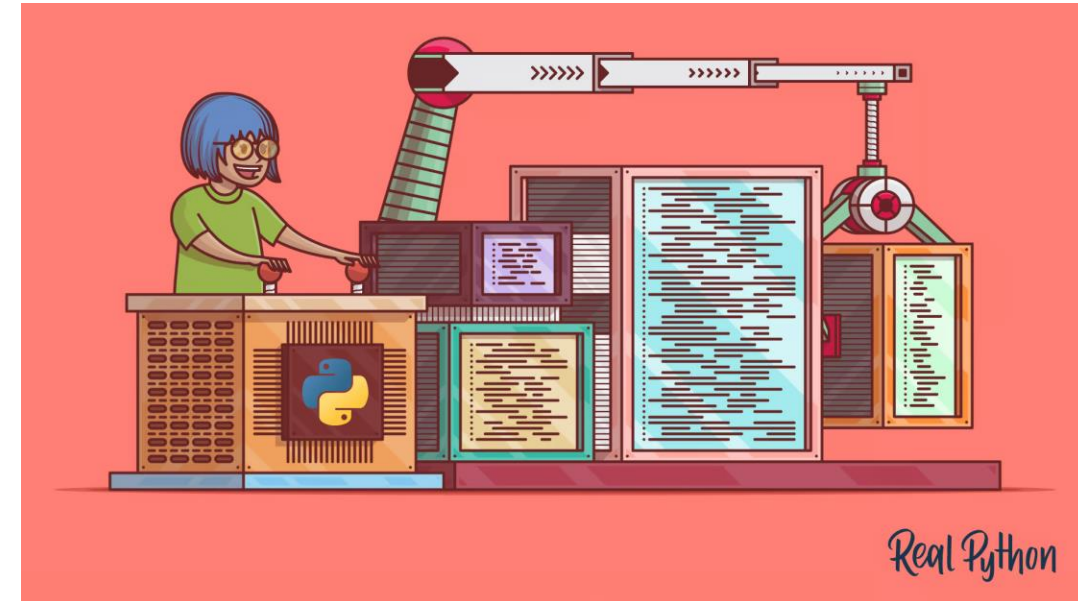
Structuring large(r) projects

Fulvio Corno

Giuseppe Averta

Carlo Masone

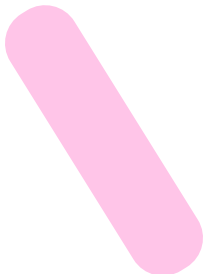
Francesca Pistilli



<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
- The `import` statement runs the module source and makes the definitions available
- The defined names are created in a separated *namespace* to avoid confusion

## Standard library modules

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

## User-defined modules

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

## Downloaded modules

- From <https://pypi.org/> - over 500k projects
- Install using pip

# 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)

```
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`
  - Example: `import math`; use `math.sin(math.pi)`
- `import module_name as alt_name`
  - 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

# 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

```
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', ...]
```

# 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...
- 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

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

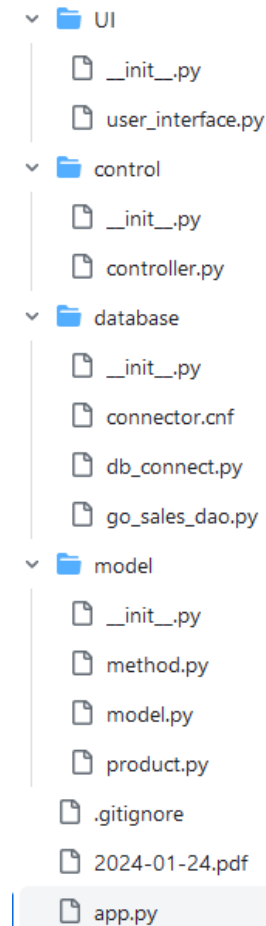
```
if __name__ == "__main__":  
    v = voto_casuale()  
    print(repr(v))
```

- Or, better:

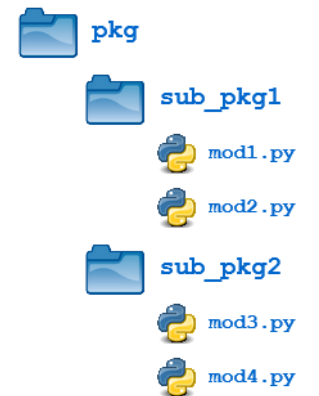
```
def _main():  
    v = voto_casuale()  
    print(repr(v))  
  
if __name__ == "__main__":  
    _main()
```

# 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**



(and sub-packages)



# Importing from packages

- The traditional syntax still applies
  - `import pkg.mod`
  - `from pkg.mod import name`
  - `from pkg.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`

# 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

# 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

# Installing packages with PyCharm

The screenshot shows the PyCharm interface for installing a Python package. The 'Python Packages' tool window is open, displaying a search for 'mariadb'. The search results are divided into 'Installed (0 found)' and 'PyPI (9 found)'. The 'mariadb' package is selected, and the 'Install' button is highlighted. The package details for 'mariadb' are shown on the right, including the version 'latest' and the 'Install package' button. The MariaDB logo is also visible.

Callouts in the image:

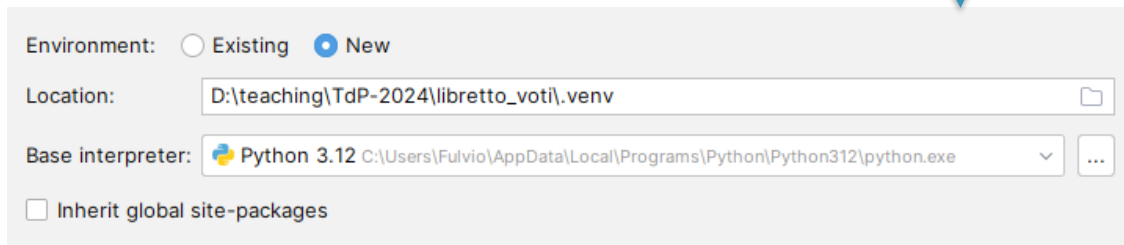
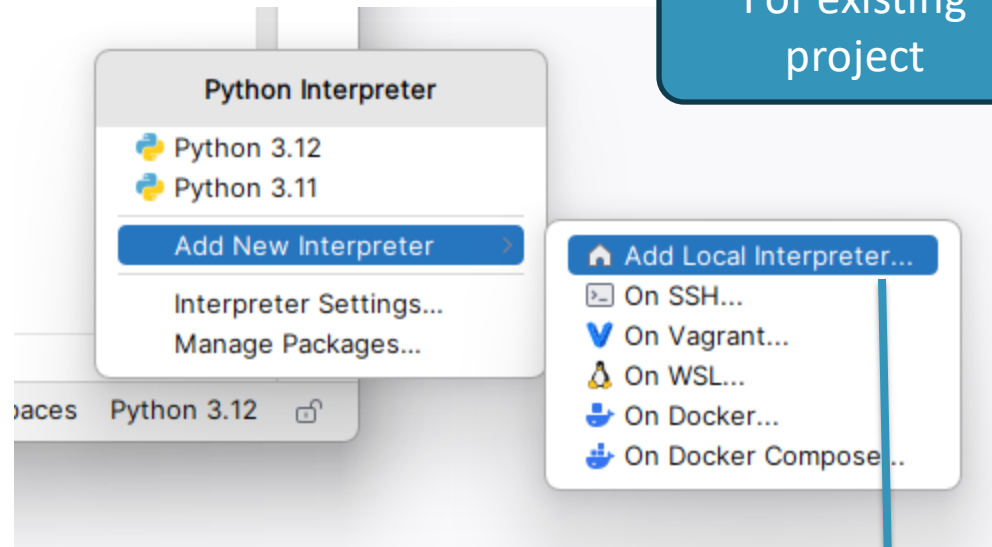
- Search Package**: Points to the search bar containing 'mariadb'.
- Currently installed**: Points to the 'Installed (0 found)' section.
- Python Packages**: Points to the left sidebar of the tool window.
- Install**: Points to the 'Install package' button.
- Info**: Points to the MariaDB logo and name.

# Virtual Environments

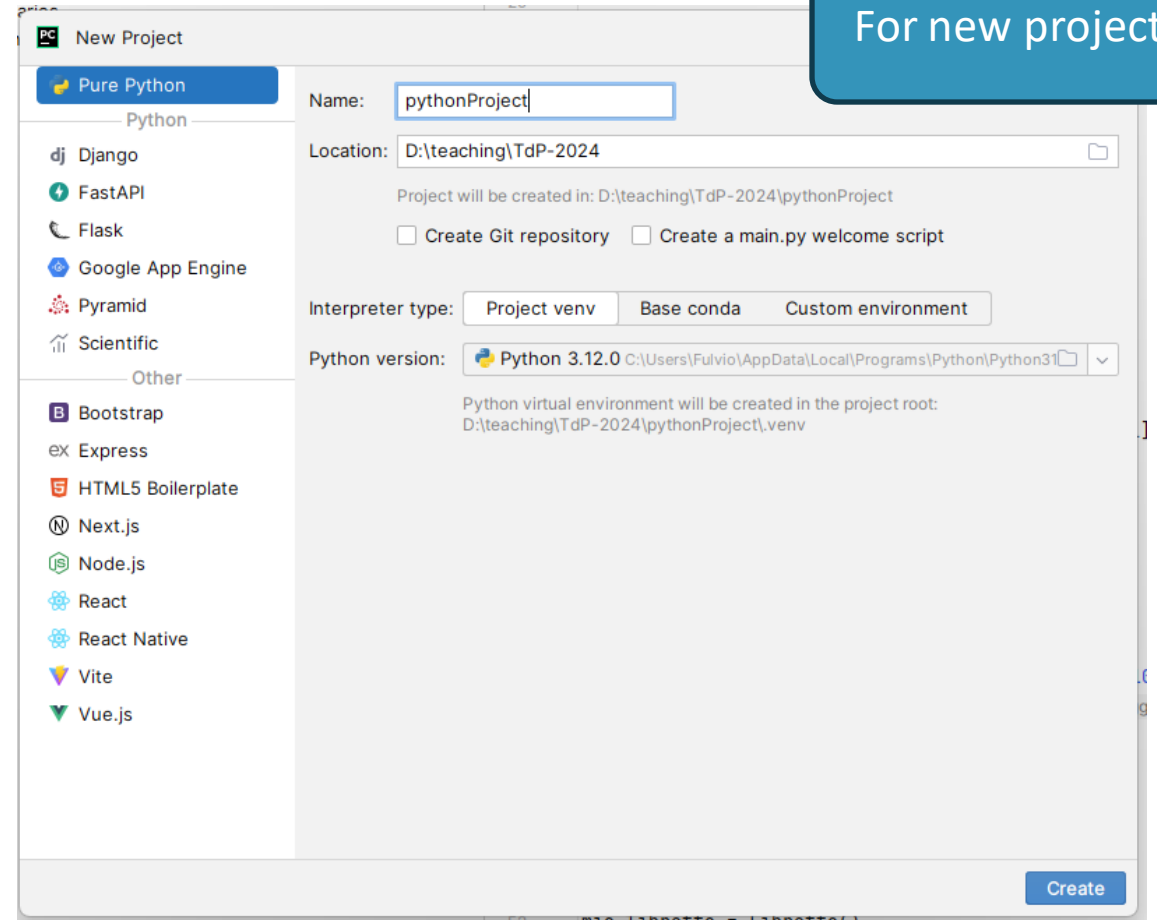
- Different projects may require different packages
- ⚠️ Your local Python library will contain all sorts of packages, that are used by some project
- ⚠️ When shipping a project, it's not clear which packages are needed to run it
- Python has a mechanism for separating the packages needed by each project
- 💡 Virtual environments
- It's a local “copy” of the Python interpreter, alongside with the packages needed for that project
- Stored in the `.venv` directory

# Virtual Environments in PyCharm

For existing project



For new project





# 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
ftpy',
'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_matplotli
b_backend', 'D:\\teaching\\TdP-2024\\libretto_voti']
```

# requirements.txt

- 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/>

