The structure of a Python project

Introduction

A Python project may initially consist of only one or two files. In this case, you don't have to worry about a project structure. But the situation is different if the program grows. In addition to the actual Python code files, documentation could also be added at some point. And as a rule, the tests should be stored in separate files and a specially designated directory. Inevitably, the question of a meaningful structure arises.

But what is the best structure for a Python project? This question is not easy to answer. There are different suggestions on the Internet, as well as in specialist literature. The following statements are also only to be understood as a suggestion. They may serve as a starting point for your projects.

A simple project structure

A typical structure for a project named `my_project` could look like this:

```
.
├── README.md
├── docs
├── my_project
│   ├── __init__.py
│   └── main.py
├── pyproject.toml
├── pytest.ini
├── requirements-dev.txt
```

```
├── requirements.txt
├── setup.cfg
├── setup.py
└── tests
    └── __init__.py
```

There are three subfolders within `my_project`:

- `my_project`: This folder has (here) the same name as the project. Instead of the project name, this folder is sometimes also called `src`. This is a package, which is why this folder contains a file called `__init__.py`. This file is usually empty (nowadays). It is called when a package or a module of this package is imported.
- `docs`: This directory contains all the files required for the documentation (e.g. for the documentation generator Sphinx).
- `tests`: All files containing tests are located here. This directory also has a file called `__init__.py`. If pytest is used, a configuration file for pytest can optionally be added to the project folder: `pytest.ini`.

The external libraries required for a project, for example numpy or matplotlib, are listed in the `requirements.txt` file. They can then be installed using `pip`:

```
pip3 install -r requirements.txt # Linux, macOS
pip install -r requirements.txt  # Windows
```

Some developers store the packages that are not immediately required for the execution of this project in a file called `requirements-dev.txt` (e.g. `pytest`). However, this is optional.

Now let's look at the configuration files `setup.py`, `setup.cfg` and `pyproject.toml`. The `setup.py` file should no longer be used in new projects, as it is now marked as *deprecated*. Nevertheless, it is often part of older projects.

Nowadays, the file `pyproject.toml` is *the* configuration file. The file `setup.cfg` may also be present. These files allow you to create an installation package. They are required, for example, if a Python program is to be published on the platform PyPi (so that it can be installed via `pip`). However, `setup.cfg` is not absolutely necessary.

Previously, the `setup.py` file contained the metadata about the project. Now this is in the `pyproject.toml` file. Or the metadata is divided into the files `pyproject.toml` and `setup.cfg`. Now the question arises, why or when do you need the configuration file

`setup.cfg`? Well, `setup.cfg` can play a role when the `setuptools` are used as build tools. Furthermore, many older programs use `setup.cfg`, and migration to `pyproject.toml` may be a step-by-step process.

In summary, if your project is completely switched to `pyproject.toml` and all metadata (name, version, dependencies, etc.) and build instructions are contained there, you no longer need `setup.cfg`.

pyroject.toml and setup.cfg in one project

If a project contains both configuration files, it is because, for example, the `setup.cfg` file contains the metadata and the `pyproject.toml` file only contains the information about the build system. This will be the case for older projects that have not yet completely switched to `pyproject.toml`.

The `setup.cfg` file could then have the following content:

```
[metadata]
name = my_project
version = 1.0.0
author =
author_email =
url =
description =
long_description = file: README.md
long_description_content_type = text/markdown
license = MIT
license_file = LICENSE
requires_python = >=3.12
classifiers =
    License :: OSI Approved :: MIT License
    Operating System :: OS Independent
    Programming Language :: Python :: 3
    Programming Language :: Python :: 3.12
    Programming Language :: Python :: 3.13
    Programming Language :: Python :: Implementation :: CPython
```

And the `pyproject.toml` file contains information about the build system to use. The following example uses `setuptools` as build system:

```
[build-system]
requires = ["setuptools>=70.0"]
build-backend = "setuptools.build_meta"
```

In addition to `setuptools`, `Hatchling`, `Flit` and `PDM` can also be used. You can find more information about how to package a Python project and examples of the build tools on this website.

Project that only uses the pyproject.toml file

As already mentioned, the sole presence of the `pyproject.toml` file is sufficient. This will be the case for new projects.[1] Below is an example of the contents of this configuration file. You can find more on this topic on the website mentioned above. (The example shown comes from that website.)

```
[project]
name = "example_package_YOUR_USERNAME_HERE"
version = "0.0.1"
authors = [
  { name="Example Author", email="author@example.com" },
]
description = "A small example package"
readme = "README.md"
requires-python = ">=3.10"
classifiers = [
    "Programming Language :: Python :: 3",
    "Operating System :: OS Independent",
    "License :: OSI Approved :: MIT License",
]

[project.urls]
Homepage = "https://github.com/pypa/sampleproject"
Issues = "https://github.com/pypa/sampleproject/issues"
```

[1] An example of this is the package manager uv. It creates a simple project structure that contains only the pyproject.toml file (and no other configuration files).

Additional notes

The file `main.py` represents the entry point of the program. The name can be chosen freely, but often this file is called `main.py` or `cli.py` (if it's a console program).

It's a good idea to add a `README.md` file to each project. If you want to publish a project or if you use Github (or a similar service), this file is required. This is where other developers can find useful information about the project. In addition, projects (on Github or GitLab, etc.) often also contain a `CONTRIBUTING.md` file that offers other developers tips on how to participate in the project.

Take a look at `setup.cfg` and `pyproject.toml` files of other projects on Github. This gives you an idea of what kind of metadata others have added.

In this example the `tests` folder is located at the top directory level. But other developers prefer to make this directory a subdirectory of the package directory.

The logging module enables the implementation of a logging system. Configuration is done via a file named `logging.ini`. An example of the structure of this configuration file can be found in this project.

There may be other configuration files. If **mypy** is used, the file `mypy.ini` may be present. If **flake8** is used, the file `.flake8` may be present. The `.flake8` file begins with a dot. On Unix-like systems (Linux, macOS) this indicates a hidden file. It is therefore not displayed in the file manager.

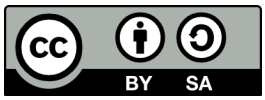
On Github you can find a shell script that can be used to automatically create a project structure. This script does not claim to be complete. But it may be a good starting point for your own script. I've also published a simple program written in C on GitHub that can be used to create a new project. Additionally, there are package managers like uv that can be used to set up a new Python project. If you're interested in this topic, you should take a look at my blog post.

As mentioned above, the structure presented here is a suggestion. More complex Python applications may require the project to be structured differently. For example, there are programs that can be started via the terminal but also have a graphical user interface. In this case it makes sense to use different subdirectories for this functionality.

License

- Version: 2.1 — June 10th, 2025
- Author: Bodo Schönfeld
- Homepage: https://niftycode.eu
- Github: https://github.com/niftycode

Attribution 4.0 International (CC BY 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 for any purpose, even commercially