The structure of a Python project

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
├── 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.

Finally, there are two files remaining: `setup.py` and `setup.cfg`. In this project example you can see both of these files. But that doesn't have to be the case. In other projects there may only be one `setup.py` file.

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`).

There used to be only one file called `setup.py`. If only this file is used (and not `setup.cfg` as well), it contains the basic information about the project. For our sample project, a `setup.py` file could have the following content:

```
from setuptools import setup, find_packages

VERSION = '0.1.0'

setup(
    name='my_project',
    version=VERSION,
    license='MIT',
    description='Just an example.',
    author='My Name',
    author_email='my_name@xyz.com',
```

```
    url='https://github.com/<user>/<project>',
    packages=find_packages(exclude=('tests', 'docs')),
    python_requires='>=3.11'
)
```

The `setup.py` file can become quite complex. For this reason, the configuration file `setup.cfg` was introduced. If this file is used, the content of `setup.py` is reduced to two lines:

```
import setuptools

setuptools.setup()
```

And instead of `setup.py` the `setup.cfg` file contains all of the required metadata:

```
[metadata]
name = my_project
author =
author-email =
license =
long_description = file: README.md
url = https://
requires-python = >= 3.11
```

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. Not least if the project is to be published, a file with this name 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 `setup.py` 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.

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.

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 are called in the terminal but also have a graphical user interface. In this case it makes sense to use different subdirectories for this functionality.

License