# Python Packaging Essentials

Joseph Barbier

2025-04-06

# Table of contents

# Preface

The aim of this site it to provide all the must known practices when it comes to create a Python package. It offers 10 blog posts, where each of them covers one topic with a few key points. The goal here is to empower anyone with just basic Python knowledge.

We'll over concrete examples, use clear explanations, and try as much as possible to go straight to the point.

### Good to know

- All blog posts are independant. Even if they follow some sort of order, it's perfectly fine to just look at what interests you.

# 1 Introduction to packaging

This is a book created from markdown and executable code.

# 2 Organize a package

In order to follow the following steps, you need to have both Git and uv installed on your machine. Both of them are CLI. This means that using your terminal, you'll be able to run commands that do things.

Let's assume we name our Python package *"sunflower"*.

## 2.1 Initialize the directory

The very first step, it to create a new directory named *"sunflower"*. In this directory, we create another directory, also named *"sunflower"*. This will looks like this:

```
sunflower/
   sunflower/
```

## 2.2 Add main Python project files

We need to create a few new files, that will be at the root of the project.

### 2.2.1 `pyproject.toml`

All the metadata of the package. It will contain many info useful when we'll want to distribute this package PyPI so that everyone can install it easily. Here is a simple version of this file.

```toml
[project]
name = "sunflower"
version = "0.1.0"
description = "Add your description here"
readme = "README.md"
requires-python = ">=3.13"
authors = [
  { name="author_name", email="email" },
```

```
]
dependencies = []
classifiers = [
    "Programming Language :: Python :: 3",
    "License :: OSI Approved :: MIT License",
    "Operating System :: OS Independent",
]

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

### 2.2.2 LICENSE

A basic text file that contains the license of your package. This is important because it tells other people what they are allowed to do with your package. This is specific to every project, but you can learn more at choosealicence.com. Here is a example of the most common license: the MIT license.

```
Copyright (c) 2025 Your Name

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

### 2.2.3 `.git/`

This is a directory used internally by the Git software to track every changes in the project. Assuming the first *"sunflower"* directory is in your `Desktop/` directory, you need to create this directory by running the `git init` command when you are in the `Desktop/sunflower/` directory. It's very likely that will **not see it**, as most operating systems (Windows, MacOS, etc) hide files/directories that start with a "`.`", but it does not matter.

### 2.2.4 `.gitignore`

A file where each line describes one or multiple files/directories that are not explicitely part of the project or not relevant in general. Don't worry too much about it, you can just start with the example content below. It's very likely that will **not see it**, as most operating systems (Windows, MacOS, etc) hide files/directories that start with a "`.`", but it does not matter.

```
# Python-generated files
__pycache__/
*.py[oc]
build/
dist/
wheels/
*.egg-info

# Virtual environments
.venv/
venv/
.env/
env/

# VS code config
.vscode/

# files on mac
.DS_Store

# all cache files
*cache*

# Sandbox files
sandbox.py
sandbox.ipynb
```

### 2.2.5 `.venv/`

A directory that contains all the things we need to properly work in our Python environment. It contains a Python interpreter, all the packages used in the project (e.g `numpy`, `requests`, etc), and a few other things. The best way to create one is to run `uv venv`. It's very likely that will **not see it**, as most operating systems (Windows, MacOS, etc) hide files/directories that start with a "`.`", but it does not matter.

### 2.2.6 `README.md`

A markdown file that describes the project, give hints on how to use it, how to install etc. There is no rule on what to do with this file, it's just used to tell people what's the first thing they should read before using your package. For instance, it could be something like this:

```
# sunflower: my cool Python package

Welcome to the homepage of the `sunflower` project.

It's a new project, but it will be available soon!
```

### 2.2.7 `sandbox.py`

A file that we'll use to test and use our package. It's facultative but very convenient.

As you can see, we haven't written a single line of Python code, and yet we already have lots of files and directories. Now our project organization looks like this:

```
sunflower/
  sunflower/
  .git/
  .venv/
  .gitignore
  README.md
  LICENSE
  sandbox.py
  pyproject.toml
```

## 2.3 Add Python code

When creating a Python package, we want to write **reusable piece of code**, and not just put some scripts that do things. To illustrate:

### 2.3.1 Not reusable code

```
name = "Joseph"
message = f"Hello {name}"
print(message)
```

```
Hello Joseph
```

This code here actually does something: it prints a message.

### 2.3.2 Reusable code

```
def say_hello(name):
   message = f"Hello {name}"
   print(message)
```

This code above does "nothing". The only thing it does is that it creates a function object that will be stored in memory. Now I can call it and it will execute code. For example:

```
say_hello("Joseph")
```

```
Hello Joseph
```

Here we'll make it simple and assume we'll only want to want to provide functions and classes into our package.

Now, let's create our first Python module (which is just a file ending with `.py`). We'll name it `module1.py`, but it could be anything we want. The only thing we want to respect when naming files are:

- use lowercase only
- avoid spaces and odd characters
- keep it short
- use underscores "_"

### 2.3.3 Bad file names

```
my file.py
Myfile.py
myFile.py
my@file.py
my-file.py
this-file-does-this-and-that.py
```

### 2.3.4 Good file names

```
my_file.py
myfile.py
```

So let's put our `module1.py` in `sunflower/sunflower/`, which will lead to:

```
sunflower/
   sunflower/
       module1.py
   .git/
   .venv/
   .gitignore
   README.md
   LICENSE
   sandbox.py
   pyproject.toml
```

In `module1.py`, we'll add our very first function in our package. For instance, we'll create a `count_sunflower()` function. This function will count how many times the word "sunflower" is in a given string.

```
import re

def count_sunflowers(s):
   s = re.sub(r"[^a-zA-Z\s]", "", s)  # Remove non-text characters
   s = s.lower()  # Convert to lowercase
   n_sunflower = s.split().count("sunflower")
   n_sunflowers = s.split().count("sunflowers")
   return n_sunflower + n_sunflowers
```

Now we'll add a `__init__.py` file at the same place as the `module1.py` file that contains our previous function. This is a special file in Python. What it does is that it tells Python that the `sunflower/sunflower/` directory is a package, which will allow us to import functions from this package in the outside world.

We now have this:

```
sunflower/
   sunflower/
       __init__.py
       module1.py
   .git/
   .venv/
   .gitignore
   README.md
   LICENSE
   sandbox.py
   pyproject.toml
```

For the moment, the `__init__.py` file must looks like this:

```python
from .module1 import count_sunflowers

__all__ = ["count_sunflowers"]
```

And congrats! You might have not realized it, but we now already have a usable Python package with 1 function in it.

## 2.4 User perspective

Now let's see how to use our package from the user perspective.

Once again, we'll need to run a command in our terminal at `Desktop/sunflower/`:

```
uv pip install -e .
```

This command will install our current package in editable mode. This enables us to test our package while making updates to it.

The next step is to open `sandbox.py` from earlier and write some code that uses our package.

```
from sunflower import count_sunflower

text = """
Sunflower petals bright and gold,
Sunflower fields, a sight to behold.
Sunflower dreams in the morning light,
Blooming softly, pure and bright.
"""

print(count_sunflower(text))
```

3

Note that we can use all of the following syntaxes:

### 2.4.1 Syntax 1

```
from sunflower import count_sunflower

text = """
Sunflower petals bright and gold,
Sunflower fields, a sight to behold.
Sunflower dreams in the morning light,
Blooming softly, pure and bright.
"""

count_sunflower(text)
```

### 2.4.2 Syntax 2

```
import sunflower

text = """
Sunflower petals bright and gold,
Sunflower fields, a sight to behold.
Sunflower dreams in the morning light,
Blooming softly, pure and bright.
"""

sunflower.count_sunflower(text)
```

### 2.4.3 Syntax 3

```python
import sunflower as sfl # or any other alias like "sf" or "sunflo"

text = """
Sunflower petals bright and gold,
Sunflower fields, a sight to behold.
Sunflower dreams in the morning light,
Blooming softly, pure and bright.
"""

sfl.count_sunflower(text)
```

## 2.5 Final organization

```
sunflower/
  sunflower/
      __init__.py
      module1.py
      module2.py
      utils.py
  .git/
  .venv/
  .gitignore
  README.md
  LICENSE
  sandbox.py
  pyproject.toml
```

# 3 Handling dependencies

This is a book created from markdown and executable code.

# 4 Unit tests

This is a book created from markdown and executable code.

# 5 Writing documentation

This is a book created from markdown and executable code.

# 6 Errors and warnings

This is a book created from markdown and executable code.

# 7 Github Actions

This is a book created from markdown and executable code.

# 8 Pre-commit

This is a book created from markdown and executable code.

# 9 API design

This is a book created from markdown and executable code.

# 10 Publish to PyPI

This is a book created from markdown and executable code.