from py-pkgs import CHEAT SHEET

① Packaging Tools

This Python Packaging cheat sheet will help you build a Python Package in no time! It relies on the following requirements:

Tool	Description
poetry	Python dependency & package management tool
GitHub	Online project management & code version control system
cookiecutter	Directory & file templating tool

② Package Structure

Python packages have a standard structure. *poetry* and *cookiecutter* can help you set up this structure quickly with the following steps:

1. Install *cookiecutter* (if it is not installed):

```
$ pip install cookiecutter
```

2. Run the UBC-MDS *cookiecutter* template and follow the prompts:

```
$ cookiecutter https://github.com/UBC-
MDS/cookiecutter-ubc-mds.git
```

3. Change into the root directory of your new package (here named "*mypkg*") :

```
$ cd mypkg
```

4. Initialize a *poetry* project:

```
$ poetry init
```

You should end up with a directory structure similar to that shown below. If you're after a package with a command line interface (CLI), see the CLI package chapter of py-pkgs.

```
mypkg
├── CONDUCT.rst
├── CONTRIBUTING.rst
├── docs
├── mypkg
├── .gitignore
├── .github
├── LICENSE
├── pyproject.toml
├── README.md
└── tests
```

③ Write Your Code

Once your Python package is set up, you can start writing your code! Your package may consist of functions, classes, a command line interface, or anything other Python code you wish!

④ Tests

Your package should also contain tests to verify that code is working as expected. *pytest* is an easy to use testing framework, with a typical workflow of:

1. Add *pytest* as a development dependency:

```
$ poetry add --dev pytest
```

2. Write tests in `mypkg/tests/test_mypkg.py`. Guidelines for writing tests can be found here, but they typically look something like this:

```
def test_myfunc():
    assert mypkg.myfunc(1, 5) == 6
    assert mypkg.myfunc(-1, -5) == -6
```

3. Run tests and make sure they are passing:

```
$ poetry run pytest
```

4. Calculate test coverage with *pytest-cov*:

```
$ poetry add --dev pytest-cov
$ poetry run pytest --cov=mypkg tests/
```

⑤ Documentation

The UBC-MDS *cookiecutter* template provides basic package documentation, such as a README, LICENSE, CONDUCT, CONTRIBUTING file and a populated docs folder.

You will still need to write documentation for your code as necessary, including:

1. Inline comments;
2. Block comments;
3. Docstrings.

Documentation can be rendered using *sphinx* and *sphinxcontrib-napoleon*:

1. Add these tools as package dependencies:

```
$ poetry add --dev sphinx
sphinxcontrib-napoleon
```

2. Render docstrings into documentation if required:

```
$ poetry run sphinx-apidoc -f -o
docs/source mypkg
```

3. Render package documentation in `docs`:

```
$ cd docs
$ poetry run make html
```

4. Upload to Read the Docs following these instructions if desired.

⑥ Releasing

Your package should ideally adopt the semantic versioning scheme, e.g., *v0.1.0*. For help implementing versioning, deprecation, or the release process in general, see the Releasing and Versioning chapter of py-pkgs. Releasing will typically involve the following:

1. Bump package version if required:

```
$ poetry version patch/minor/major
```

2. Ensure tests are passing:

```
$ poetry run pytest
```

3. Build package

```
$ poetry build
```

4. Release to TestPyPI and check you can install your package.

```
$ poetry config repositories.test-pypi
https://test.pypi.org/legacy/
$ poetry publish -r test-pypi
$ pip install --index-url
https://test.pypi.org/simple/ --extra-
index-url https://pypi.org/simple
mypkg
```

5. If all is working as expected, release to PyPI:

```
$ poetry publish
```

py-pkgs

⑦ CI/CD

There are many tools available for implementing CI/CD for your package. In py-pkgs, we advocate for GitHub Actions.

The UBC-MDS *cookiecutter* used in Step 2 provides an option for including pre-mase CI and/or CD workflow files in your package structure, which can be modified as desired and are triggered when you push your package repository to GitHub. Take a look at the CI/CD chapter of py-pkgs for more information.

⑧ Acknowledgments

This cheat sheet was inspired by the Rstudio Cheatsheets. Thanks also to Cookiecutter and Jupyter Book for providing the open-source frameworks to build and support py-pkgs.