

# MANAGING SOFTWARE AND PACKAGING IN PYTHON

**Aina Frau-Pascual**  
**ASPP summer school 2017**

# INDEX

1. Organize your code into libraries and modules that can be imported and reused
2. Create a Python package that can be installed with pip
3. Distribute your package
4. Upload it to PyPI
5. Versioning and tagging in git

# 1) ORGANIZE YOUR CODE INTO LIBRARIES AND MODULES THAT CAN BE IMPORTED AND REUSED

Minimal structure:

```
example/  
    __init__.py  
    example_file.py
```

example\_file.py > contains a function example\_fun()

# 1) ORGANIZE YOUR CODE INTO LIBRARIES AND MODULES THAT CAN BE IMPORTED AND REUSED

`LICENSE.md`: Open source license MIT, Apache 2.0, and GPLv3... <https://choosealicense.com/>, <https://opensource.guide/legal/>

`README.md`: explain how to use your project...

`CONTRIBUTING.md`: Contributing guidelines

# 1) ORGANIZE YOUR CODE INTO LIBRARIES AND MODULES THAT CAN BE IMPORTED AND REUSED

It is good to have a `.gitignore` file containing the files that we do not want to commit to git:

```
# Compiled python modules.  
*.pyc  
  
# Setuptools distribution folder.  
/dist/  
  
# Python egg metadata, regenerated  
# from source files by setuptools.  
/*.*egg-info
```

# EXERCISE:

- a) Create your own package:
  - i) Remember to add your `__init__.py`

## 2) CREATE A PYTHON PACKAGE THAT CAN BE INSTALLED WITH PIP

If you want to install it with pip you need a setup.py file.

The directory structure should look like

```
example/  
  example/  
    __init__.py  
    example_file.py  
  setup.py
```

## 2) CREATE A PYTHON PACKAGE THAT CAN BE INSTALLED WITH PIP

And setup.py should look like:

```
from setuptools import setup

setup(name='example',
      version='0.1',
      description='The ultimate example',
      url='http://github.com/author/example',
      author='Author Name',
      author_email='author@example.com',
      license='MIT',
      packages=['example'],
      zip_safe=False)
```



## 2) CREATE A PYTHON PACKAGE THAT CAN BE INSTALLED WITH PIP

We can install the package locally (for use on our system)

```
pip install -e .
```

with “-e” the installation is "editable".

Now you can import it from anywhere in the system

# EXERCISE:

- a) Create your own package:
  - i) Remember to add your `__init__.py`
- b) Make it pip installable
  - i) Create your `setup.py`
  - ii) Install the package locally

### 3) DISTRIBUTE YOUR PACKAGE

Distribution Package: versioned archive file that contains Python packages, modules, and other resource files that are used to distribute a Release.

Repositories: PyPI and conda-forge

- The Python Package Index (PyPI): manage with pip
- Conda-forge: manage with conda

## 3) DISTRIBUTE YOUR PACKAGE

Distribution formats: source vs built

- Source Distribution (or “sdist”): requires a build step before it can be installed:
  - `setuptools`: for creating and installing distributions. Enhancement to `sdist`.
- Built Distribution: only needs to be moved to the correct location on the target system, to be installed.
  - `Wheel`: Introduced by PEP 427. Currently supported by `pip`.

## 4) UPLOAD/PUBLISH IT TO PYPI: USING TWINE

Register in <https://pypi.org/>  
<https://testpypi.python.org/pypi>

Create some distributions in the normal way

```
pip install wheel  
python setup.py sdist bdist_wheel
```

Upload with twine

```
twine upload --repository-url  
https://test.pypi.org/legacy/ dist/*
```

(\*) if we do not give a `--repository-url`, the default is PyPI

## 4) UPLOAD/PUBLISH IT TO PYPI: INSTALL FROM PYPI

Now you can install the package with

```
pip install --index-url  
https://test.pypi.org/simple/ example
```

(\*) if we do not give a `--index-url`, the default is PyPI

or with wheels

```
pip install -use-wheel example  
pip install example.whl
```

# EXERCISE:

- a) Create your own package:
  - i) Remember to add your `__init__.py`
- b) Make it pip installable
  - i) Create your `setup.py`
  - ii) Install the package locally
- c) Upload to PyPI
  - i) Register to test-PyPI
  - ii) Upload your package

## 5) PROPER VERSIONING AND TAGGING IN GIT

Git has the ability to tag specific points in history as being important: release points, publications...



## 5) PROPER VERSIONING AND TAGGING IN GIT

Git has the ability to tag specific points in history as being important: release points, publications...

```
git tag v1.4-lw
```

```
git tag -a v1.4 -m "my tag"
```

```
git tag -a v1.2 9fceb02
```

```
git tag
```

```
git tag -l "v1.8.5*"
```

lightweight tag: pointer to a specific commit.

Annotated tags: stored as full objects in the Git database.

To tag past commits specifying the commit checksum (or part of it)

lists tags in alphabetic order

lists specific tags

## 5) PROPER VERSIONING AND TAGGING IN GIT

To see the existing tags

To push tags

or many at once

To checkout tags

To fetch tags created  
from github

```
git show v1.2
```

```
git push origin v1.5
```

```
git push origin --tags
```

```
git checkout -b version2 v2.0.0
```

```
git fetch
```

## 5) PROPER VERSIONING AND TAGGING IN GIT

How can I tag them?

```
git tag v2017.08
```

Date

```
git tag v1.4.1
```

Release

```
git tag NI-17/08/30-v1
```

Publication

```
git tag jon
```

Any name

# EXERCISE:

- a) Create your own package:
  - i) Remember to add your `__init__.py`
- b) Make it pip installable
  - i) Create your `setup.py`
  - ii) Install the package locally
- c) Upload to PyPI
  - i) Register to test-PyPI
  - ii) Upload your package
- d) Commit changes to git and add a tag

# ANY QUESTIONS?

Some sites for extended information:

<https://github.com/pypa/sampleproject>

<https://python-packaging.readthedocs.io>

<https://packaging.python.org/>

## 0.A) UPLOAD TO CONDA-FORGE TO BE INSTALLED WITH CONDA

Create recipe for a new package

- Fork conda-forge/staged-recipes
- Create new branch and a new folder in /recipes
  - Example recipe: staged-recipes/recipes/example/
- Create recipe from staged-recipes/recipes/example/
- Make a pull request
- Your package will be uploaded to conda-forge when the PR is merged

## 0.B) ENVIRONMENTS: VIRTUALENV AND CONDA

virtualenv is a tool to create isolated Python environments.

```
pip install virtualenv
virtualenv my_env
source my_env/bin/activate
deactivate
```

You can also create environments with conda

```
conda create -n myenv python=3.4
conda install -n myenv scipy
source activate my_env
source deactivate
```