Project Structure

`myproject/` ← This is your git repository, and should usually match the name on Github

Project Structure

```
myproject/
   README.md
```

← Markdown-formatted file describing the package. You write this in plain text and GitHub renders it

Markdown Resources:

- http://www.markdowntutorial.com/
- https://guides.github.com/pdfs/markdown-cheatsheet-online.pdf

Project Structure

```
myproject/
   README.md
   LICENSE
```

← The license lets you specify how others may use your code.

Licensing Resources:

- http://www.astrobetter.com/blog/2014/03/10/the-whys-and-hows-of-licensing-scientific-code/

Project Structure

```
myproject/
   README.md
   LICENSE
   myproject/
```

← The python package. When you do "`import myproject`", this is what gets imported.

Project Structure

```
myproject/
   README.md
   LICENSE
   myproject/
      __init__.py
```

← This file marks the directory as a Python package.

Project Structure

```
myproject/
   README.md
   LICENSE
   myproject/
      __init__.py
      core.py
```

← other files in the package contain importable code.

Project Structure

```
myproject/
   README.md
   LICENSE
   myproject/
      __init__.py
      core.py
      submodule/
         __init__.py
         script.py
```

← modules can have submodules (and sub-submodules, etc.) to any depth.

Project Structure

```
myproject/
   README.md
   LICENSE
   myproject/
      __init__.py
      core.py
     +submodule/
```

Project Structure

```
myproject/
   README.md
   LICENSE
   myproject/
      __init__.py
      core.py
     +submodule/
      tests/
         __init__.py
         test_core.py
```

← “unit tests” go in their own submodule

Project Structure

`__init__.py`

```
__version__ = "1.0"
from .core import add
```

`core.py`

```
def add(a, b):
    return a + b
```

`test_core.py`

```
from mymodule import add


def test_add(a, b):
    assert add(1, 1) == 2
    assert add(0, 1) == 1
    assert add(-1, 1) == 0
```

You can use, for example, pytest[1] to run these tests:

```
# in the top-level directory,
# run this shell command:
$ pytest mymodule
```

[1] http://doc.pytest.org/

Project Structure

```
myproject/
   README.md
   LICENSE
   myproject/
      __init__.py
      core.py
     +submodule/
     +tests/
```

Project Structure

```
myproject/
   README.md
   LICENSE
  +myproject/
```

Project Structure

```
myproject/
   README.md
   LICENSE
  +myproject/
   setup.py
```

← the setup.py script allows the package to be installed

`setup.py` resources:

- https://github.com/pypa/sampleproject/
- https://github.com/uwescience/shablona

Project Structure

```
myproject/
   README.md
   LICENSE
  +myproject/
   setup.py
  +doc/
```

← the documentation directory lives beside your project. One good option is sphinx[1]

[1] http://www.sphinx-doc.org/

Project Structure

```
myproject/
   README.md
   LICENSE
  +myproject/
   setup.py
  +doc/
   examples/
      example_script.py
      Demo.ipynb
```

← if you have example code or notebooks, use an examples directory.

Project Structure

```
myproject/
   README.md
   LICENSE
  +myproject/
   setup.py
  +doc/
  +examples/
```

Project Structure

```
myproject/
   README.md
   LICENSE
  +myproject/
   setup.py
  +doc/
  +examples/
  +paper/
```

← if I'm developing the code for research, I usually put the paper LaTeX here (along with scripts for figures, etc.)

Project Structure

```
myproject/
   README.md
   LICENSE
  +myproject/
   setup.py
  +doc/
  +examples/
  +paper/
   .gitignore
```

`.gitignore` example:

```
# Python bytecode
*.pyc

# notebook temporary files
.ipynb_checkpoints

# emacs temporary files
*~
```

← this file helps keep your directory clean, by telling git types of files to not track.

Project Structure

```
myproject/
   README.md
   LICENSE
  +myproject/
   setup.py
  +doc/
  +examples/
  +paper/
   .gitignore
   .travis.yml
```

← this script tells github to run your unit tests after every commit!

(See Getting Started with TravisCI)