UPPSALA
UNIVERSITET

Best Practices II: Testing, Documenting And Packaging Of Code

Day 4

Advanced Scientific Programming with Python

pytest

Adapted from:
https://jacobian.org/writing/getting-started-with-pytest/

Testing

- Testing is crucial to high quality software!
- It's important to make testing as automated as possible to make sure it's done!
- Python as several packages that provide **test frameworks**: `unittest`, `nose/nose2`, `py.test`
- There are also services that automatically build your code and run tests every time you `push` your code.
- A few of these **Continuous Integration** services are GitLab (gitlab.com), CircleCI (circleci.com) and CodeShip (codeship.com) as well as GitHub through GitHub Actions (https://github.com/features/actions).
- In this course we'll focus on `py.test` and **GitHub Actions**.

py.test in action

```python
# fib.py
def fib(n):
    """Return the first Fibonacci number above n."""
    a = 0
    b = 1
    while b < n:
        a, b = b, a + b
    return b
```

- Lets test our code above. The file with the tests should start with `test_`:

```python
# test_fib.py
import fib

# the name of the testing function should
# also start with test_
def test_fib():
    assert fib(0) == 1
```

- Now to run the test we simply do:

```
$ py.test
```

py.test in action

```
$ py.test
============================= test session starts ==============================
platform darwin -- Python 3.5.1, pytest-2.9.0, py-1.4.31, pluggy-0.3.1
rootdir: /Users/filipe/Documents/Teaching/Advanced Scientific Programming with Python/
python-course/day4-bestpractices-2/code/pytest, inifile:
collected 1 items

test_fib.py F

=================================== FAILURES ===================================
_____________________________________ test_fib _____________________________________

    def test_fib():
>       assert fib(0) == 1
E       TypeError: 'module' object is not callable

test_fib.py:4: TypeError
=========================== 1 failed in 0.01 seconds ===========================
```

As you can see you can also have bugs in your tests!

py.test in action

- And after fixing the silly error:

```
$ py.test
============================= test session starts ==============================
platform darwin -- Python 3.5.1, pytest-2.9.0, py-1.4.31, pluggy-0.3.1
rootdir: /Users/filipe/Documents/Teaching/Advanced Scientific Programming with Python/
python-course/day4-bestpractices-2/code/pytest, inifile:
collected 1 items

test_fib.py .

=========================== 1 passed in 0.01 seconds ===========================
```

- This is the most important feature of **`py.test`**, but it can do much more!
- Check pytest.org and py.test --help for more information.
- Lets do a slightly more advanced example now…

py.test in action

- Multiple test parameters with one function:

```
# test_fib_params.py
import fib
import pytest

# Fibonacci Sequence
# 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...
@pytest.mark.parametrize("n, expected", [
    (0, 1),
    (1, 2),
    (3, 5)
])
def test_fib_parametrized(n, expected):
    assert fib.fib(n) == expected
```

- Lets try this again:

```
$ py.test
```

py.test in action

```
$ py.test
============================= test session starts ==============================
platform darwin -- Python 3.5.1, pytest-2.9.0, py-1.4.31, pluggy-0.3.1
rootdir: /Users/filipe/Documents/Teaching/Advanced Scientific Programming with Python/python-
course/day4-bestpractices-2/code/pytest, inifile:
collected 4 items

test_fib.py .
test_fib_params.py .FF

=================================== FAILURES ===================================
_____________________________ test_fib_parametrized[1-2] _____________________________

n = 1, expected = 2

    @pytest.mark.parametrize("n, expected", [
        (0, 1),
        (1, 2),
        (3, 5)
    ])
    def test_fib_parametrized(n, expected):
>       assert fib.fib(n) == expected
E       assert 1 == 2
E        +  where 1 = <function fib at 0x107876840>(1)
E        +    where <function fib at 0x107876840> = fib.fib

test_fib_params.py:11: AssertionError
```

py.test in action

```
_____________________________ test_fib_parametrized[3-5] ______________________________

n = 3, expected = 5

    @pytest.mark.parametrize("n, expected", [
        (0, 1),
        (1, 2),
        (3, 5)
    ])
    def test_fib_parametrized(n, expected):
>       assert fib.fib(n) == expected
E       assert 3 == 5
E        +  where 3 = <function fib at 0x107876840>(3)
E        +    where <function fib at 0x107876840> = fib.fib

test_fib_params.py:11: AssertionError
====================== 2 failed, 2 passed in 0.02 seconds ======================
```

py.test in action

Conclusion: Our "simple" function does not do what it claims!

What does it really do...?

```
# fib.py
def fib(n):
    """Return the first Fibonacci number above n."""
    a = 0
    b = 1
    while b < n:
        a, b = b, a + b
    return b
```

Do you test your code?

socrative

UU1

- No, I was not aware of testing tools
- No, my code does not need testing
- No, it is a waste of time to write tests
- Yes, with py.test
- Yes, in some other way

GitHub Actions

Using GitHub Actions

- I've placed our Fibonacci repository, with `fib.py`, `test_fib.py` and `test_fib_param.py` on GitHub (https://github.com/uu-python/fib)
- GitHub Actions are controlled by YAML files in the .github/workflows of your repository. In this case I created a `.github/workflows/ci.yml` file with the following content:

```yaml
name: Test fib
on: [push]

jobs:
  build:
    runs-on: ubuntu-latest
    strategy:
      matrix:
        python-version: ["3.7", "3.8", "3.9", "3.10"]

    steps:
      - uses: actions/checkout@v3
      - name: Set up Python ${{ matrix.python-version }}
        uses: actions/setup-python@v4
        with:
          python-version: ${{ matrix.python-version }}
      - name: Install dependencies
        run: |
          pip install pytest
      - name: Test with pytest
        run: |
          pytest
```

Using GitHub Actions

main

Commits on Mar 7, 2023

First CI attempt
FilipeMaia committed 1 minute ago

Add python code
FilipeMaia committed 13 minutes ago

Some checks haven't completed yet
3 successful and 1 in progress checks

- ✓ Test fib / build (3.7) (push) Successful in 9s Details
- ✓ Test fib / build (3.8) (push) Successful in 7s Details
- ✓ Test fib / build (3.9) (push) Successful in 8s Details
- Test fib / build (3.10) (push) *In progress — This ch...* Details

Using GitHub Actions

uu-python / fib Public

Edit Pins | Unwatch 3 | Fork 0 | Star 0

Code | Issues | Pull requests | Actions | Projects | Wiki | Security | Insights | Settings

← Test fib

First CI attempt #1

Re-run all jobs

Summary

Jobs

- build (3.7)
- build (3.8)
- build (3.9)
- build (3.10)

Run details

- Usage
- Workflow file

build (3.7)
succeeded 6 minutes ago in 9s

Search logs

Set up job 2s

Run actions/checkout@v3 1s

```
1  ▶ Run actions/checkout@v3
12 Syncing repository: uu-python/fib
13 ▶ Getting Git version info
17 Temporarily overriding HOME='/home/runner/work/_temp/adeba563-0aec-46f5-a092-736f6560d029' before making global git config changes
18 Adding repository directory to the temporary git global config as a safe directory
19 /usr/bin/git config --global --add safe.directory /home/runner/work/fib/fib
20 Deleting the contents of '/home/runner/work/fib/fib'
21 ▶ Initializing the repository
35 ▶ Disabling automatic garbage collection
37 ▶ Setting up auth
43 ▶ Fetching the repository
65 ▶ Determining the checkout info
66 ▶ Checking out the ref
70 /usr/bin/git log -1 --format='%H'
71 'aaa08c3b4f3932728ef8ccda3945da18be69bc56'
```

Set up Python 3.7 0s

Install dependencies 5s

Test with pytest 0s

Post Set up Python 3.7 0s

Post Run actions/checkout@v3 0s

Complete job 0s

Demo: Show Github Actions Output

Using GitHub Actions

- You can find documentation specific to Python testing on https://docs.github.com/en/actions/automating-builds-and-tests/building-and-testing-python
- There is also a Python "starter workflow" file at https://github.com/actions/starter-workflows/blob/main/ci/python-package.yml, which is a great starting point for further editing.

A more complex example

```yaml
# https://github.com/FXIhub/hummingbird/blob/master/.github/workflows/test.yml
# This workflow will install Python dependencies, run tests and lint with a single version of Python
# For more information see: https://help.github.com/actions/language-and-framework-guides/using-python-with-github-actions

name: test

on:
  push:
    branches: [ master ]
  pull_request:
    branches: [ master ]

jobs:
  dummy-build:

    runs-on: ubuntu-latest
    strategy:
      matrix:
        mpi: ['mpi4py', 'none']
        python-version: [2.7, 3.7, 3.8]
    steps:
    - uses: actions/checkout@v2
    - uses: conda-incubator/setup-miniconda@v2
      with:
        python-version: ${{ matrix.python-version }}
    - name: Install base dependencies
      shell: bash -l {0}
      run: |
        # $CONDA is an environment variable pointing to the root of the miniconda directory
        # $CONDA/bin/conda env update --file environment.yml --name base
        # Install requirements except for h5writer and pint which must be from pip
        pip install Pint
        conda install --yes h5py pexpect pyqtgraph scipy pyzmq tornado pytz pyqt
        pip install h5writer
```

A more complex example

```yaml
    - name: Install mpi
      shell: bash -l {0}
      run: conda install --yes ${{ matrix.mpi }}
      if: matrix.mpi != 'none'
    - name: Lint with flake8
      shell: bash -l {0}
      run: |
        conda install flake8
        # stop the build if there are Python syntax errors or undefined names
        # $CONDA/bin/flake8 . --count --select=E9,F63,F7,F82 --show-source --
statistics
        # exit-zero treats all errors as warnings. The GitHub editor is 127 chars wide
        # $CONDA/bin/flake8 . --count --exit-zero --max-complexity=10 --max-line-
length=127 --statistics
    - name: Install subprocess32
      shell: bash -l {0}
      run: |
        # subprocess32 is necessary for testing under python2.7
        conda install subprocess32
      if: matrix.python-version == 2.7
    - name: Test with pytest
      shell: bash -l {0}
      run: |
        conda install pytest
        pytest
```

A more complex example

```
lcls-build:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v2
    - name: Create data dir
      shell: bash -l {0}
      run: mkdir data
    - name: Cache data
      uses: actions/cache@v2
      with:
        path: ~/data
        key: xtc-home-data
        restore-keys: xtc-home-data
    - name: Prepare data
      shell: bash -l {0}
      run: |
        mkdir -p ${HOME}/data
        cd ${HOME}/data
        [ -f e41-r0073-s00-c00.xtc ] && echo "File already exists" || wget http://davinci.icm.uu.se/wp-content/
uploads/xtc/e41-r0073-s00-c00.xtc
        [ -f e41-r0092-s00-c00.xtc ] && echo "File already exists" || wget http://davinci.icm.uu.se/wp-content/
uploads/xtc/e41-r0092-s00-c00.xtc
    - name: Pull image
      shell: bash -l {0}
      run: |
        docker pull filipemaia/hummingbird-testing
    - name: LCLS Test Run
      shell: bash -l {0}
      run: |
        docker run -v ${GITHUB_WORKSPACE}:/opt/hummingbird -v ${HOME}/data:/opt/data filipemaia/hummingbird-
testing /bin/sh -c "source /reg/g/psdm/etc/ana_env.sh && mkdir -p /reg/d/psdm/AMO/amo15010 && ln -s /opt/data /
reg/d/psdm/AMO/amo15010/xtc && /reg/g/psdm/sw/external/python/2.7.10/x86_64-rhel6-gcc44-opt/bin/pip install
pint==0.9 && cd /opt/hummingbird && /reg/g/psdm/sw/external/python/2.7.10/x86_64-rhel6-gcc44-opt/bin/coverage
run --source src -m py.test  -v --color=yes --showlocals --durations=10"
```

Continuous Integration

Do you use continuous integration in your projects?

socrative
UU1

- No, I do all checks manually
- No, my projects are too simple
- No, I don't need it
- Yes!
- Other

Coverage Testing

- Coverage testing allows you to know what parts of your code have been tested
- We'll use Codecov to do our coverage testing
- It's quite simple to setup Codecov
- First go to https://app.codecov.io/ and setup the repo you want to use.
- Then just follow the instructions

Let's get your repo covered

GitHub Actions | Other CI

Step 1: add repository token as repository secret

```
CODECOV_TOKEN=c1a92c50-74b2-4e2c-8c2f-ca21bfa21c7b
```

Step 2: configure Codecov's GitHub app

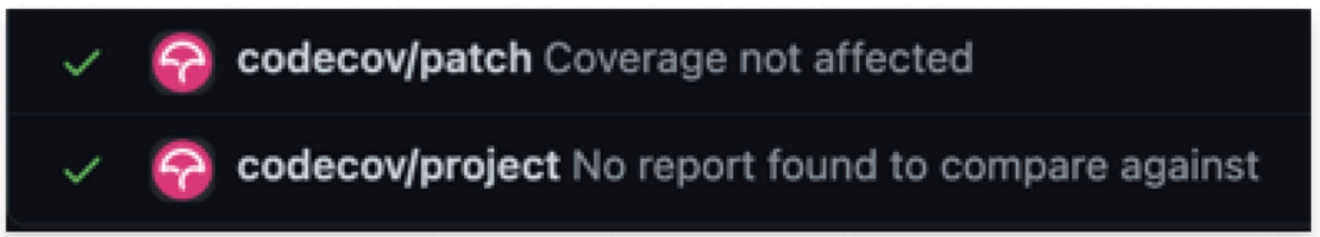
Codecov will use the integration to post statuses and comments

Step 3: add Codecov to your GitHub Actions workflow

```
- name: Upload coverage reports to Codecov
  uses: codecov/codecov-action@v3
```

Step 4: get coverage analysis from Codecov

Once you've committed your changes in step 2 and ran your CI/CD pipeline. In your pull request, you should see two status checks:



Coverage Testing

- I simply added the following to our ci.yml

```yaml
- name: Generate coverage report
  run: |
    pip install pytest-cov
    pytest --cov=./ --cov-report=xml
- name: Upload coverage reports to Codecov
  uses: codecov/codecov-action@v3
```

main

Commits on Mar 7, 2023

Update Codecov
FilipeMaia committed 6 minutes ago

Add Codecov
FilipeMaia committed 16 minutes ago

First CI attempt
FilipeMaia committed 41 minutes ago

Add python code
FilipeMaia committed 54 minutes ago

All checks have passed
6 successful checks

Test fib / build (3.9) (push) Successful in 14s Details

Test fib / build (3.10) (push) Successful in 16s Details

codecov/patch — Coverage not affected Details

codecov/project — No report found to compare agai... Details

Coverage Testing

Update Codecov

8 minutes ago FilipeMaia authored commit adeb90a ✓ CI Passed main

HEAD adeb90a	Patch	Change	Source
100.00%	–	–	

Files changed[0] Indirect changes[0] **File explorer**

Uncovered | **Partial** | Covered

Code tree File list **fib** Search for files

Files	Tracked lines	Covered	Partial	Missed	Coverage %
fib.py	6	6	0	0	100.00%
test_fib.py	3	3	0	0	100.00%
test_fib_params.py	5	5	0	0	100.00%

When should you add testing to your code?

Documentation

- Code is read more times that it's written
- To understand the code and make use of it you need **documentation**
- Python includes **docstrings** to help you document your code
- docstrings are string literal that occurs as the first statement in a module, function, class, or method definition
- All modules should normally have docstrings, and all functions and classes exported by a module should also have docstrings
- Public methods (including `__init__.py`) should also have docstrings
- A package may be documented in the module docstring of the `__init__.py` file in the package directory

Documentation

```
# fib.py
def fib(n):
    """Return the first Fibonacci number above n."""
    a = 0
    b = 1
    while b < n:
        a, b = b, a + b
    return b
```

← Here's the docstring

- You can write anything you want in a docstring but a consistent standard greatly improves their usefulness
- One such standard is **numpydoc**
- You can read about it in at:
- https://numpydoc.readthedocs.io/en/latest/format.html#docstring-standard

Numpydoc

```
def source(object, output=sys.stdout):
    """
    Print or write to a file the source code for a Numpy object.

    The source code is only returned for objects written in Python. Many
    functions and classes are defined in C and will therefore not return
    useful information.

    Parameters
    ----------
    object : numpy object
        Input object. This can be any object (function, class, module,
        ...).
    output : file object, optional
        If `output` not supplied then source code is printed to screen
        (sys.stdout).  File object must be created with either write 'w' or
        append 'a' modes.

    See Also
    --------
    lookfor, info

    Examples
    --------
    >>> np.source(np.interp)                        #doctest: +SKIP
    In file: /usr/lib/python2.6/dist-packages/numpy/lib/function_base.py
    def interp(x, xp, fp, left=None, right=None):
        \"\"\".... (full docstring printed)\"\"\"
        if isinstance(x, (float, int, number)):
            return compiled_interp([x], xp, fp, left, right).item()
        else:
            return compiled_interp(x, xp, fp, left, right)

    The source code is only returned for objects written in Python.

    >>> np.source(np.array)                         #doctest: +SKIP
    Not available for this object.
    """
```

- Here's a typical numpydoc docstring (in this case for **`numpy.source`**)
- The section names and separators are standardised.
- As all well as the parameters section.

Sphinx

Python documentation generator

Adapted from:
https://codeandchaos.wordpress.com/2012/07/30/sphinx-autodoc-tutorial-for-dummies/
http://www.sphinx-doc.org/en/stable/tutorial.html

Sphinx

- Sphinx can be used to turn your docstrings into beautiful web pages
- It was originally created for the Python documentation, and it has excellent facilities for the documentation of software projects in a range of languages.
- First lets make proper numpydoc docstrings:

```
def fib(n):
    """
    Return the first Fibonacci number above n.

    Iteratively calculate Fibonacci numbers until it finds one
    greater than n, which it then returns.

    Parameters
    ----------
    n : integer
      The minimum threshold for the desired Fibonacci number.

    Returns
    -------
    b : integer
      The first Fibonacci number greater than the input, `n`.

    Examples
    --------
    >>> fib.fib(1)
    2
    >>> fib.fib(3)
    5
    """
```

- Now lets setup our Sphinx configuration:

```
$ mkdir docs
$ sphinx-quickstart
Welcome to the Sphinx 1.2.3 quickstart utility.

Please enter values for the following settings (just press Enter to
accept a default value, if one is given in brackets).

Enter the root path for documentation.
> Root path for the documentation [.]: docs
…
```

- I took mostly the default options except I enabled autodoc and asked for a Makefile to be created
- Now we need to tell it where our code is. Edit the newly created **docs/conf.py** and add:

```
sys.path.insert(0,os.path.abspath('..'))
```

- We also need to add **numpydoc** to the list of extension:

```
extensions = [
    'sphinx.ext.autodoc',
    'numpydoc'
]
```

Sphinx

- And finally tell Sphinx the modules that should be documented. Edit **`index.rst`** to add the modules of interest:

```
Welcome to fibonacci's documentation!
=====================================

Contents:

.. toctree::
   :maxdepth: 2

.. automodule:: fib
   :members:

Indices and tables
==================

* :ref:`genindex`
* :ref:`modindex`
* :ref:`search`
```

- To actually generate the documentation run:

```
$ make html
sphinx-build -b html -d _build/doctrees   . _build/html
Making output directory...
Running Sphinx v1.2.3
loading pickled environment... not yet created
building [html]: targets for 1 source files that are out of date
updating environment: 1 added, 0 changed, 0 removed
reading sources... [100%] index
looking for now-outdated files... none found
pickling environment... done
checking consistency... done
preparing documents... done
writing output... [100%] index
writing additional files... genindex py-modindex search
copying static files... done
copying extra files... done
dumping search index... done
dumping object inventory... done
build succeeded.

Build finished. The HTML pages are in _build/html.

$ open _build/html/index.html
```

Table Of Contents

This Page

Show Source

Quick search

Go

Enter search terms or a module, class or function name.

Welcome to fibonacci's documentation!

Contents:

`fib.`**`fib`**(*n*)

Return the first Fibonacci number above n.

Iteratively calculate Fibonacci numbers until it finds one greater than n, which it then returns.

Parameters:	**n** : integer The minimum threshold for the desired Fibonacci number.
Returns:	**b** : integer The first Fibonacci number greater than the input, *n*.

Examples

```
>>> fib.fib(1)
2
>>> fib.fib(3)
5
```

Indices and tables

- *Index*
- *Module Index*
- *Search Page*

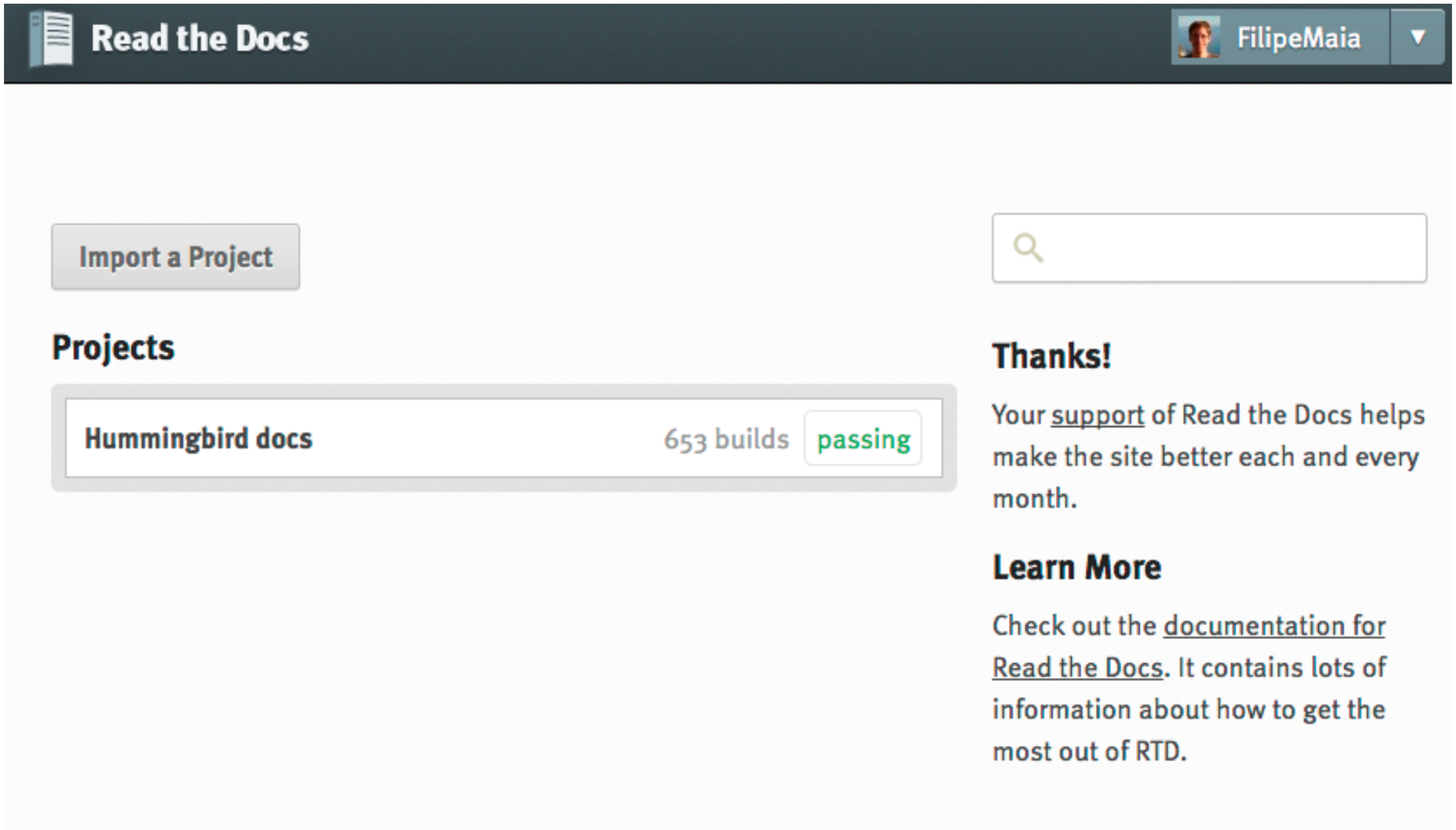
How do you document your code?

- It's all in my head!
- With comments in the code
- With a PDF file
- With a documentation system (e.g. Sphinx)
- I will document it when the project is finished

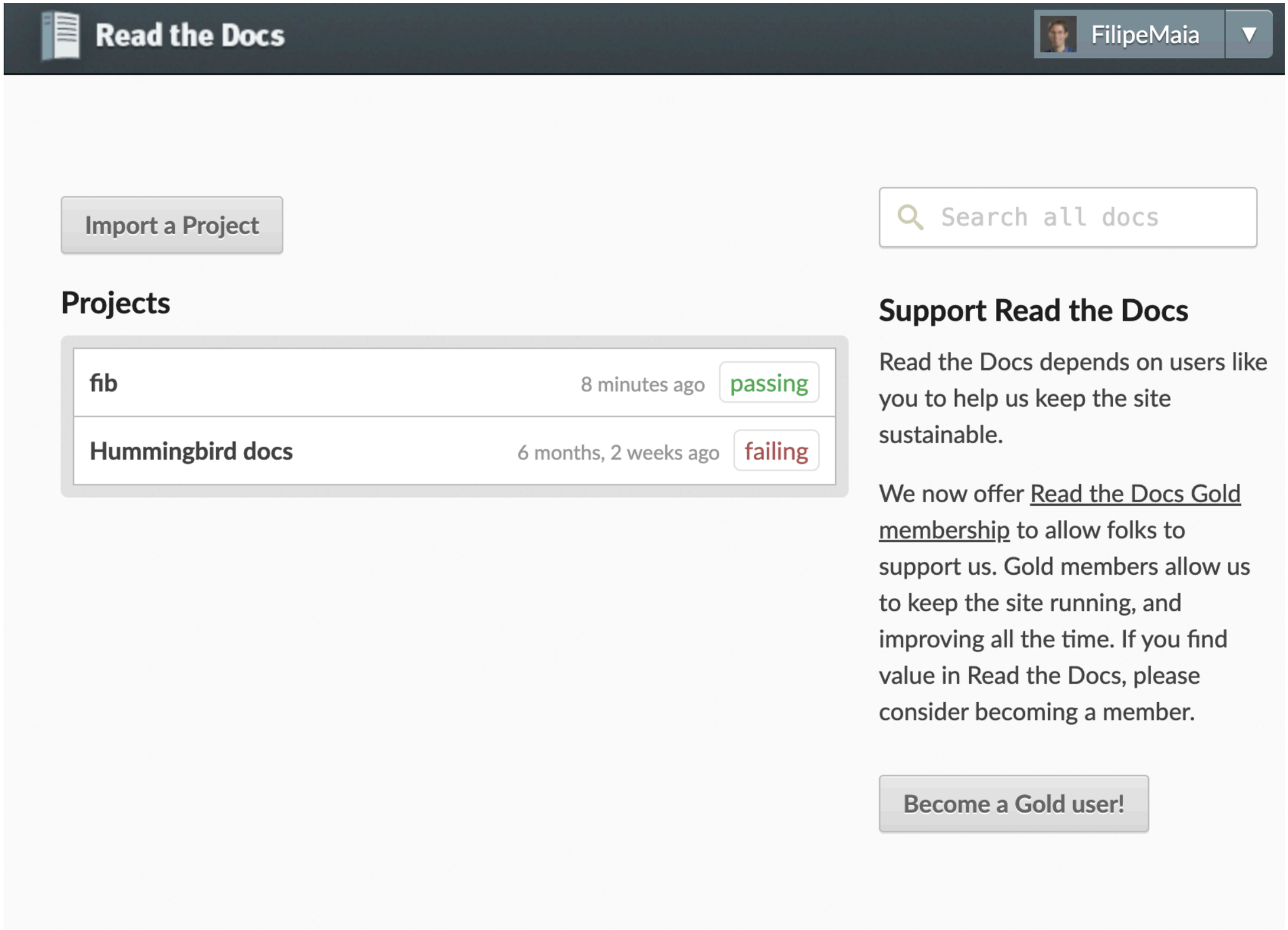
Adapted from:
http://docs.readthedocs.io/en/latest/getting_started.html

Read The Docs

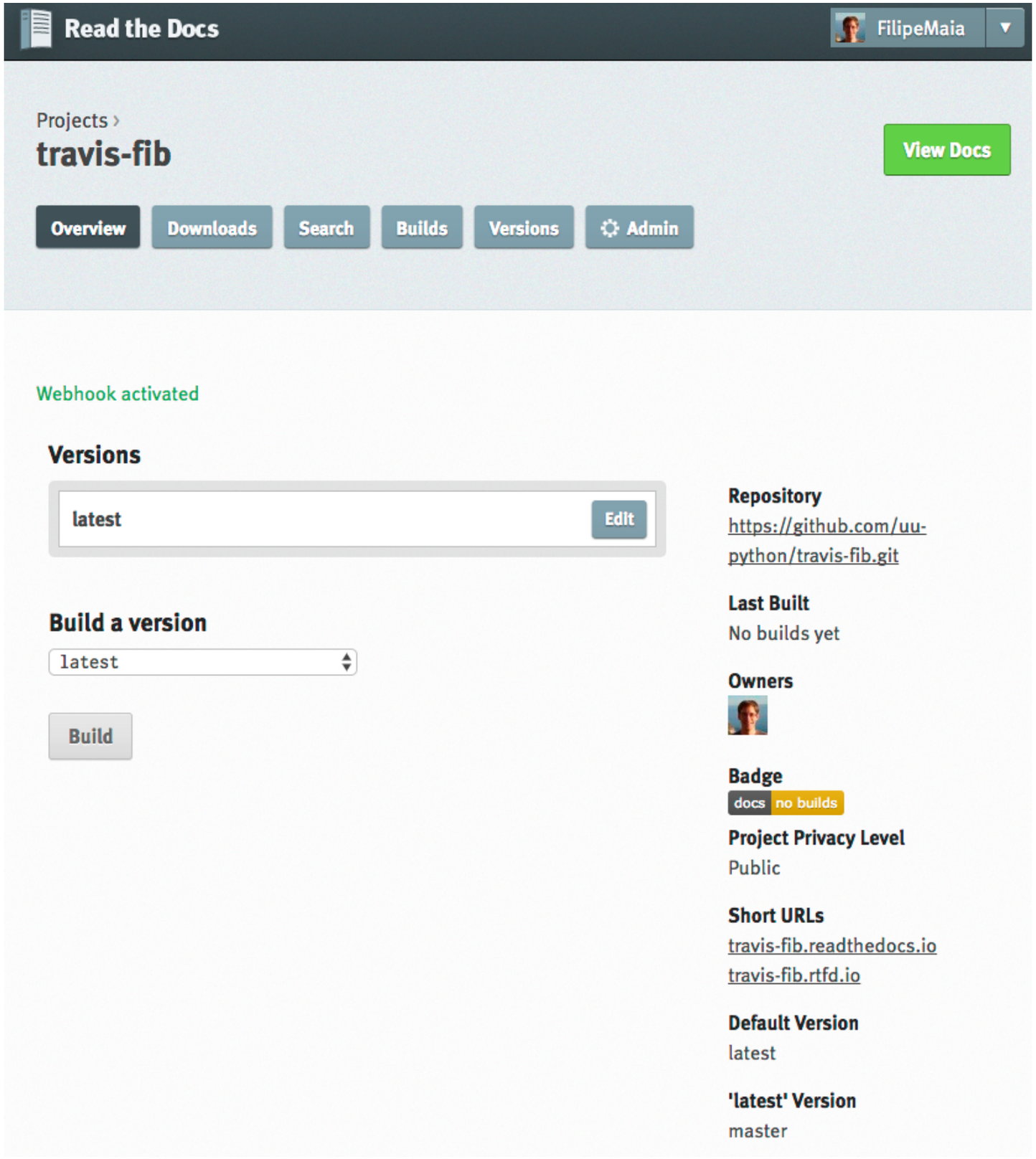
- Read The Docs (https://readthedocs.org) makes it easy to put your Sphinx documentation online.
- It's as simple as creating an account, connecting to your GitHub account and importing the project.



Read The Docs



Read The Docs



Read The Docs

- I needed to create a **`requirements.txt`** file in the root of the package with a single line saying "**`numpydoc`**".
- This lets Read The Docs know that it needs to install **`numpydoc`** before trying to generate the documentation otherwise I got:

```
Running Sphinx v1.5.3
making output directory...
...
ExtensionError: Could not import extension numpydoc (exception: No module named
numpydoc)

Extension error:
Could not import extension numpydoc (exception: No module named numpydoc)
Command time: 0s Return: 1
```

- Requirement files are an important of packaging, which we'll see in the next section.

Welcome to fibonacci's documentation!

Contents:

fib.fib(*n*)

Return the first Fibonacci number above n.

Iteratively calculate Fibonacci numbers until it finds one greater than n, which it then returns.

Parameters: **n : integer**

The minimum threshold for the desired Fibonacci number.

Returns: **b : integer**

The first Fibonacci number greater than the input, *n*.

Examples

```
>>> fib.fib(1)
2
>>> fib.fib(3)
5
```

Indices and tables

- Index
- Module Index
- Search Page

https://fib.readthedocs.io/

Adapted from:
https://python-packaging.readthedocs.io/en/latest/minimal.html

Packaging

- Packaging provides an easy way for people to install your software
- PyPI (https://pypi.org/) the Python Package Index is a repository of software for the Python programming language.
- It currently (Feb 2021) contains 291 169 packages.
- Those are the packages that can be installed with `pip`.
- We'll now see how to get your small Fibonacci code into PyPI.

Picking A Name

- Python module/package names should generally follow the following constraints:

  1. All lowercase
  2. Unique on pypi, even if you don't want to make your package publicly available (you might want to specify it privately as a dependency later)

- I went with the uu-fibonacci.

- The initial file structure is as follows:

```
uu-fibonacci/
    docs/
    fib.py
    requirements.txt
```

- Now we have to create the `setup.py` file

setup.py

```
# setup.py
from setuptools import setup

setup(name='uu-fibonacci',
      version='1.0',
      description='Example package that calculates fibonacci numbers',
      url='https://github.com/uu-python/travis-fib',
      author='Filipe Maia',
      author_email='filipe.c.maia@gmail.com',
      license='BSD',
      py_modules=['fib'],
      packages=[])
```

- Now you can already install the package locally:

```
$ pip install . --user
Processing /Users/filipe/Documents/Teaching/Advanced Scientific Programming with
Python/python-course/day4-bestpractices-2/code/uu-fibonacci
Installing collected packages: uu-fibonacci
  Running setup.py install for uu-fibonacci ... done
Successfully installed uu-fibonacci-1.0
```

Publishing on PyPI

- First lets create a distribution file we can upload

```
$ python setup.py sdist
…
$ ls dist/
uu-fibonacci-1.0.tar.gz

$ twine register dist/uu-fibonacci-1.0.tar.gz
Registering package to https://pypi.python.org/pypi
Registering uu-fibonacci-1.0.tar.gz

$ twine upload dist/uu-fibonacci-1.0.tar.gz
Uploading distributions to https://pypi.python.org/pypi
Uploading uu-fibonacci-1.0.tar.gz
[==============================] 3592/3592 - 00:00:01
```

Testing our Package

```
$ pip uninstall uu-fibonacci
Uninstalling uu-fibonacci-1.0:
  /Users/filipe/Library/Python/2.7/lib/python/site-packages/fib.py
  /Users/filipe/Library/Python/2.7/lib/python/site-packages/fib.pyc
  /Users/filipe/Library/Python/2.7/lib/python/site-packages/uu_fibonacci-1.0-
py2.7.egg-info
Proceed (y/n)? y
  Successfully uninstalled uu-fibonacci-1.0

$ pip install uu-fibonacci --user
Collecting uu-fibonacci
  Downloading uu-fibonacci-1.0.tar.gz
Installing collected packages: uu-fibonacci
  Running setup.py install for uu-fibonacci ... done
Successfully installed uu-fibonacci-1.0
```

Testing our Package

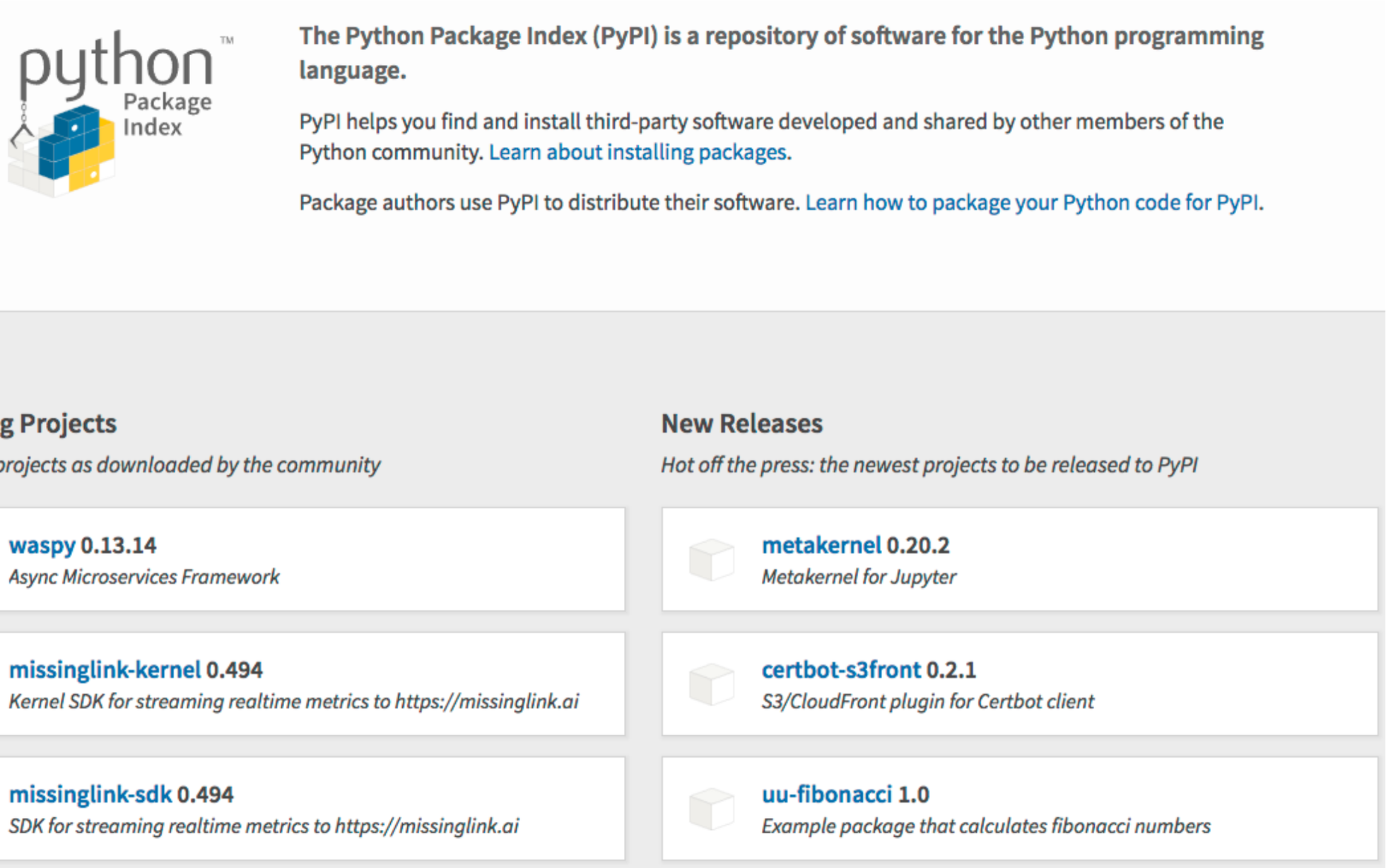
```
$ ipython2
In [1]: import fib

In [2]: fib.fib(2)
Out[2]: 3

In [3]: fib.fib?
Signature: fib.fib(n)
Docstring:
Return the first Fibonacci number above n.

Iteratively calculate Fibonacci numbers until it finds one
greater than n, which it then returns.

Parameters
----------
n : integer
  The minimum threshold for the desired Fibonacci number.

Returns
-------
b : integer
  The first Fibonacci number greater than the input, `n`.

Examples
--------
>>> fib.fib(1)
2
>>> fib.fib(3)
5
File:      ~/Library/Python/2.7/lib/python/site-packages/fib.py
Type:      function
```

It works!

More About Packaging

- There's much more to learn about packaging
- I would suggest a thorough reading of :

`https://packaging.python.org/distributing/`



More About Packaging

Do you package your code?

socrative
UU1

- Yes, so others can use it more easily
- Yes, so I can test it more easily
- Yes, so I can use it more easily
- No, it's too much trouble
- No, there's no need


ANACONDA®

Packaging With Conda

- `pip` gives you one way to package projects
- If your project gets large and complex it might not be enough
- In particular when you start depending on system libraries and non Python code you will find problems
- `conda` (the packaging system of Anaconda) tries to handle this case
- If you look inside your Anaconda/Miniconda directory you'll see a mini root filesystem:

```
[filipe:~/miniconda3] $ ls
LICENSE.txt  conda-bld   condabin  etc      lib    pkgs        share  src
bin          conda-meta  envs      include  locks  python.app  shell  ssl
```

For more information check:

https://enterprise-docs.anaconda.com/en/latest/data-science-workflows/packages/build.html

Basic Conda Recipe

somefile_recipe/

meta.yaml

```yaml
package:
  name: abc
  version: 1.2.3

source:
  path: .

requirements:
  host:
    - python
  run:
    - python
```

build.sh

```
cp somefile.py $SP_DIR
```

$SP_DIR - Python site-packages location

bld.bat

```
COPY somefile.py %SP_DIR%
```

somefile.py

```
print("yes I'm a file.  I live in {}".format(__file__))
```

```
# Building the package
$ conda install conda-build
$ conda build somefile_recipe
```

```
# Upload to anaconda.org
$ conda install anaconda-client
$ anaconda login
$ anaconda upload abc

# You can now install with:
$ conda install -c <username> abc
```

https://www.youtube.com/watch?v=xiI1i525ljE

https://github.com/python-packaging-tutorial/python-packaging-tutorial

Finding Recipes

- Existing recipes (best):
  - https://github.com/AnacondaRecipes
  - https://github.com/conda-forge
  - e.g. https://github.com/conda-forge/mpi4py-feedstock/tree/master/recipe
- Skeletons from PyPI
  - `conda skeleton pypi <package name on pypi>`
  - Read metadata from upstream repository
  - Translate that into a recipe
- If all else fails, build your own!
- Also check conda-smithy for cross-platform recipes!

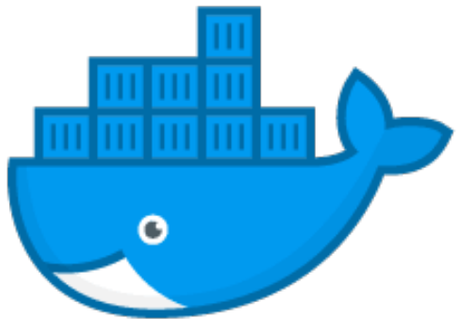
https://docs.conda.io/projects/conda-build/en/latest/index.html

Conda Demo

Containers

Containers

- Containers are another solution when you need to package an entire system!
- They contain an entire operative system, somewhat separate from the host system.
- Several option exist, the most popular of which is Docker.
  https://www.docker.com/



- More recently Lawrence Berkeley National Lab developed Singularity.
- Singularity is a container software aimed at High Performance Computing

  https://sylabs.io/docs/

When to use what packaging method?

Visualization

- Data visualisation is an important part of data analysis
- There are multiple data visualisation and plotting packages for Python.
- Among the most popular are **`plotly`**, **`plotnine`** (aka ggplot) and **`Altair`**
- Despite its many quirks the most widely used is still **`matplotlib`** which I'll briefly demonstrate.

Matplotlib Notebook

SciPy

- The scipy package contains various toolboxes dedicated to common issues in scientific computing.
- Its different submodules correspond to different applications, such as interpolation, integration, optimization, image processing, statistics, special functions, etc.
- scipy can be compared to other standard scientific-computing libraries, such as the GSL (GNU Scientific Library for C and C++), or Matlab's toolboxes.
- scipy is the core package for scientific routines in Python; it is meant to operate efficiently on numpy arrays, so that numpy and scipy work hand in hand.

Why SciPy?

- Before implementing a routine, it is worth checking if the desired data processing is not already implemented in SciPy.
- As non-professional programmers, scientists often tend to re-invent the wheel, which leads to buggy, non-optimal, difficult-to-share and unmaintainable code.
- By contrast, SciPy's routines are optimized and tested, and should therefore be used when possible.

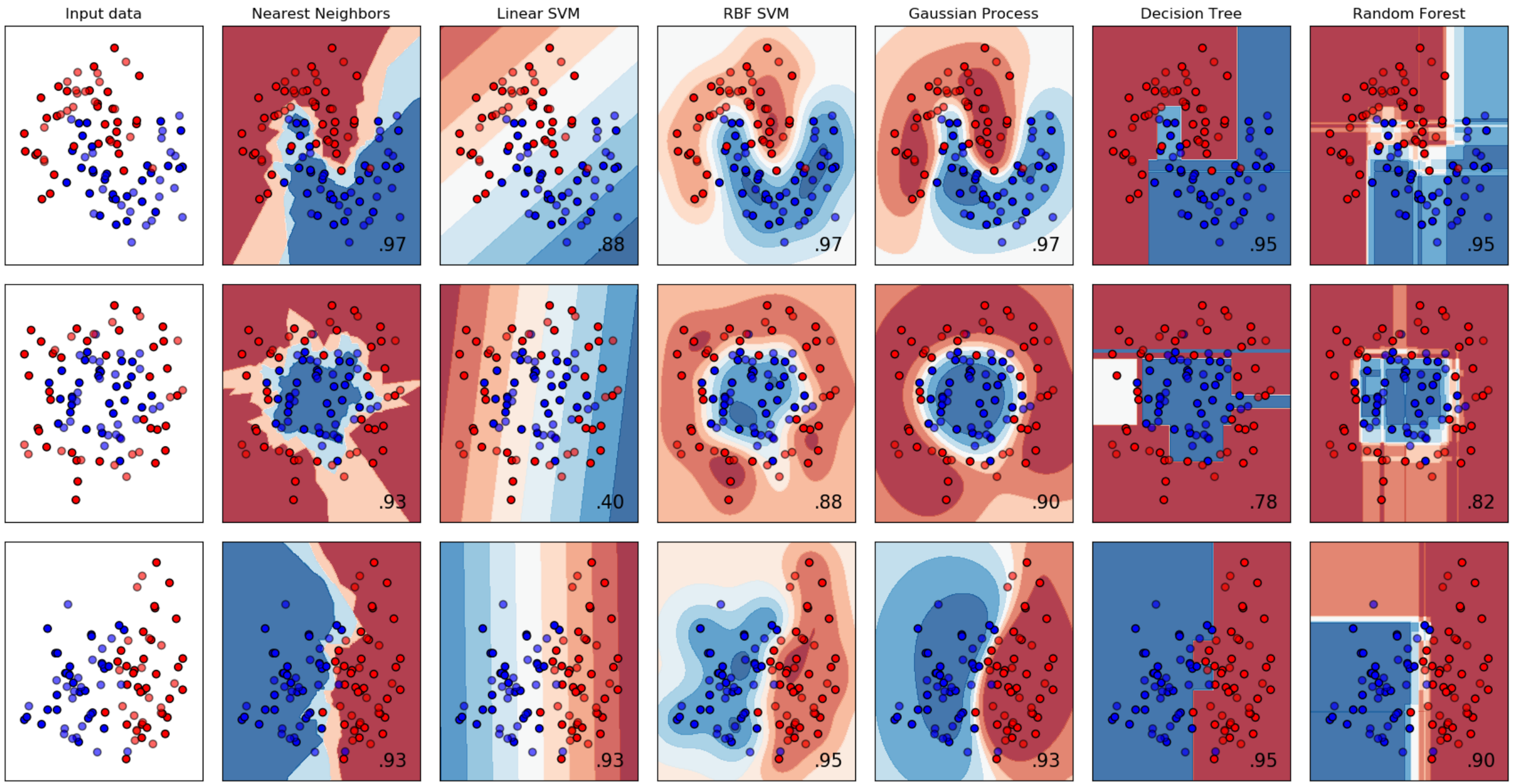
What's In SciPy?

- Clustering package (scipy.cluster)
- Constants (scipy.constants)
- Discrete Fourier transforms (scipy.fftpack)
- Integration and ODEs (scipy.integrate)
- Interpolation (scipy.interpolate)
- Input and output (scipy.io)
- Linear algebra (scipy.linalg)
- Miscellaneous routines (scipy.misc)
- Multi-dimensional image processing (scipy.ndimage)
- Orthogonal distance regression (scipy.odr)
- Optimization and root finding (scipy.optimize)
- Signal processing (scipy.signal)
- Sparse matrices (scipy.sparse)
- Sparse linear algebra (scipy.sparse.linalg)
- Compressed Sparse Graph Routines (scipy.sparse.csgraph)
- Spatial algorithms and data structures (scipy.spatial)
- Special functions (scipy.special)
- Statistical functions (scipy.stats)
- Statistical functions for masked arrays (scipy.stats.mstats)
- Low-level callback functions

SciPy Notebook

Beyond Numpy And Scipy

- scikit-learn - Simple Machine Learning in Python



Beyond Numpy And Scipy

- scikit-image - Image Processing in Python