

Starting a package

DEVELOPING PYTHON PACKAGES



James Fulton

Climate informatics researcher

Why build a package anyway?

- To make your code easier to reuse.
- To avoid lots of copying and pasting.
- To keep your functions up to date.
- To give your code to others.

Course content

You will build a full package, and cover:

- File layout
- Import structure
- Making your package installable
- Adding licenses and READMEs
- Style and unit tests for a high quality package
- Registering and publishing your package to PyPI
- Using package templates

Scripts, modules, and packages

- Script - A Python file which is run like `python myscript.py`.
- Package - A directory full of Python code to be imported
 - e.g. `numpy`.
- Subpackage - A smaller package inside a package
 - e.g. `numpy.random` and `numpy.linalg`.
- Module - A Python file inside a package which stores the package code.
 - e.g. example coming in next 2 slide.
- Library - Either a package, or a collection of packages.
 - e.g., the Python standard library (`math`, `os`, `datetime`, ...)

Directory tree of a package

Directory tree for simple package

```
mysimplepackage/  
|-- simplemodule.py  
|-- __init__.py
```

- This directory, called `mysimplepackage`, is a Python Package
- `simplemodule.py` contains all the package code
- `__init__.py` marks this directory as a Python package

Contents of simple package

`__init__.py`

Empty file

`simplemodule.py`

```
def cool_function():  
    ...  
    return cool_result  
  
...  
  
def another_cool_function():  
    ...  
    return another_cool_result
```

File with generalized functions and code.

Subpackages

Directory tree for package with subpackages

```
mysklearn/  
|-- __init__.py  
|-- preprocessing  
|   |-- __init__.py  
|   |-- normalize.py  
|   |-- standardize.py  
|-- regression  
|   |-- __init__.py  
|   |-- regression.py  
|-- utils.py
```

Let's practice!

DEVELOPING PYTHON PACKAGES

Documentation

DEVELOPING PYTHON PACKAGES



James Fulton

Climate informatics researcher

Why include documentation?

- Helps your users use your code
- Document each
 - Function
 - Class
 - Class method

```
import numpy as np
help(np.sum)
```

```
...
sum(a, axis=None, dtype=None, out=None)
    Sum of array elements over a given axis.

Parameters
-----
a : array_like
    Elements to sum.
axis : None or int or tuple of ints, optional
    Axis or axes along which a sum is performed.
    The default, axis=None, will sum all of the
    elements of the input array.

...
```

Why include documentation?

- Helps your users use your code
- Document each
 - Function
 - Class
 - Class method

```
import numpy as np
help(np.array)
```

```
...
array(object, dtype=None, copy=True)

Create an array.

Parameters
-----
object : array_like
    An array, any object exposing the array
    interface ...
dtype : data-type, optional
    The desired data-type for the array.
copy : bool, optional
    If true (default), then the object is copied.
...
```

Why include documentation?

- Helps your users use your code
- Document each
 - Function
 - Class
 - Class method

```
import numpy as np
x = np.array([1,2,3,4])
help(x.mean)
```

```
...
mean(...) method of numpy.ndarray instance
    a.mean(axis=None, dtype=None, out=None)

Returns the average of the array elements
along given axis.

Refer to `numpy.mean` for full documentation.
...
```

Function documentation

```
def count_words(filepath, words_list):  
    """  
        ...  
    """
```

Function documentation

```
def count_words(filepath, words_list):  
    """Count the total number of times these words appear."""
```

Function documentation

```
def count_words(filepath, words_list):  
    """Count the total number of times these words appear.  
  
    The count is performed on a text file at the given location.  
    """
```

Function documentation

```
def count_words(filepath, words_list):  
    """Count the total number of times these words appear.  
  
    The count is performed on a text file at the given location.  
  
    [explain what filepath and words_list are]  
  
    [what is returned]  
    """
```


Documentation style

Google documentation style

```
"""Summary line.

Extended description of function.

Args:
    arg1 (int): Description of arg1
    arg2 (str): Description of arg2
```

reStructured text style

```
"""Summary line.

Extended description of function.

:param arg1: Description of arg1
:type arg1: int
:param arg2: Description of arg2
:type arg2: str
```

NumPy style

```
"""Summary line.

Extended description of function.

Parameters
-----
arg1 : int
    Description of arg1 ...
```

Epytext style

```
"""Summary line.

Extended description of function.

@type arg1: int
@param arg1: Description of arg1
@type arg2: str
@param arg2: Description of arg2
```

NumPy documentation style

Popular in scientific Python packages like

- `numpy`
- `scipy`
- `pandas`
- `sklearn`
- `matplotlib`
- `dask`
- etc.

NumPy documentation style

```
import scipy
help(scipy.percentile)
```

```
percentile(a, q, axis=None, out=None, overwrite_input=False, interpolation='linear')
```

Compute the q-th percentile of the data along the specified axis.

Returns the q-th percentile(s) of the array elements.

Parameters

a : array_like

Input array or object that can be converted to an array.

Other types include - `int` , `float` , `bool` , `str` , `dict` , `numpy.array` , etc.

NumPy documentation style

```
import scipy
help(scipy.percentile)
```

```
percentile(a, q, axis=None, out=None, overwrite_input=False, interpolation='linear')
...
Parameters
-----
...
axis : {int, tuple of int, None}
...
interpolation : {'linear', 'lower', 'higher', 'midpoint', 'nearest'}
```

- List multiple types for parameter if appropriate
- List accepted values if only a few valid options

NumPy documentation style

```
import scipy
help(scipy.percentile)
```

```
percentile(a, q, axis=None, out=None, overwrite_input=False, interpolation='linear')
```

```
...
```

```
Returns
```

```
-----
```

```
percentile : scalar or ndarray
```

```
    If `q` is a single percentile and `axis=None`, then the result  
    is a scalar. If multiple percentiles are given, first axis of  
    the result corresponds to the percentiles...
```

```
...
```

NumPy documentation style

Other sections

- Raises
- See Also
- Notes
- References
- Examples

¹ <https://numpydoc.readthedocs.io/en/latest/format.html>

Documentation templates and style translation

- `pymment` can be used to generate docstrings
- Run from terminal
- Any documentation style from
 - Google
 - Numpydoc
 - reST (i.e. reStructured-text)
 - Javadoc (i.e. epytext)
- Modify documentation from one style to another

Documentation templates and style translation

```
pyment -w -o numpydoc textanalysis.py
```

```
def count_words(filepath, words_list):  
    # Open the text file  
    ...  
    return n
```

- `-w` - overwrite file
- `-o numpydoc` - output in NumPy style

Documentation templates and style translation

```
pyment -w -o numpydoc textanalysis.py
```

```
def count_words(filepath, words_list):  
    """  
  
    Parameters  
    -----  
    filepath :  
  
    words_list :  
  
    Returns  
    -----  
    type  
    """
```

Translate to Google style

```
pyment -w -o google textanalysis.py
```

```
def count_words(filepath, words_list):  
    """Count the total number of times these words appear.  
  
    The count is performed on a text file at the given location.  
  
    Parameters  
    -----  
    filepath : str  
        Path to text file.  
    words_list : list of str  
        Count the total number of appearances of these words.  
  
    Returns  
    -----
```

Translate to Google style

```
pyment -w -o google textanalysis.py
```

```
def count_words(filepath, words_list):  
    """Count the total number of times these words appear.  
  
    The count is performed on a text file at the given location.  
  
    Args:  
        filepath(str): Path to text file.  
        words_list(list of str): Count the total number of appearances of these words.  
  
    Returns:  
  
    """
```

Package, subpackage and module documentation

mysklearn/__init__.py

```
"""
Linear regression for Python
=====

mysklearn is a complete package for implementing
linear regression in python.
```

mysklearn/preprocessing/__init__.py

```
"""
A subpackage for standard preprocessing operations.
"""
```

mysklearn/preprocessing/normalize.py

```
"""
A module for normalizing data.
"""
```

Let's practice!

DEVELOPING PYTHON PACKAGES

Structuring imports

DEVELOPING PYTHON PACKAGES



James Fulton

Climate informatics researcher

Without package imports

```
import mysklearn
```

```
help(mysklearn.preprocessing)
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: module 'mysklearn' has no
  attribute 'preprocessing'
```

Directory tree for package with subpackages

```
mysklearn/
|-- __init__.py
|-- preprocessing
|   |-- __init__.py
|   |-- normalize.py
|   |-- standardize.py
|-- regression
|   |-- __init__.py
|   |-- regression.py
|-- utils.py
```

Without package imports

```
import mysklearn.preprocessing
```

```
help(mysklearn.preprocessing)
```

```
Help on package mysklearn.preprocessing in
mysklearn:
```

```
NAME
```

```
mysklearn.preprocessing - A subpackage
for standard preprocessing operations.
```

Directory tree for package with subpackages

```
mysklearn/
|-- __init__.py
|-- preprocessing
|   |-- __init__.py
|   |-- normalize.py
|   |-- standardize.py
|-- regression
|   |-- __init__.py
|   |-- regression.py
|-- utils.py
```


Without package imports

```
import mysklearn.preprocessing
```

```
help(mysklearn.preprocessing.normalize)
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: module
'mysklearn.preprocessing' has no attribute
'normalize'
```

Directory tree for package with subpackages

```
mysklearn/
|-- __init__.py
|-- preprocessing
|   |-- __init__.py
|   |-- normalize.py
|   |-- standardize.py
|-- regression
|   |-- __init__.py
|   |-- regression.py
|-- utils.py
```

Without package imports

```
import mysklearn.preprocessing.normalize
```

```
help(mysklearn.preprocessing.normalize)
```

```
Help on module mysklearn.preprocessing.normalize
in mysklearn.preprocessing:
```

NAME

```
mysklearn.preprocessing.normalize - A module
for normalizing data.
```

Directory tree for package with subpackages

```
mysklearn/
|-- __init__.py
|-- preprocessing
|   |-- __init__.py
|   |-- normalize.py
|   |-- standardize.py
|-- regression
|   |-- __init__.py
|   |-- regression.py
|-- utils.py
```

Importing subpackages into packages

`mysklearn/__init__.py`

Absolute import

```
from mysklearn import preprocessing
```

- Used most - more explicit

Relative import

```
from . import preprocessing
```

- Used sometimes - shorter and sometimes simpler

Directory tree for package with subpackages

```
mysklearn/  
|-- __init__.py      <--  
|-- preprocessing  
|   |-- __init__.py  
|   |-- normalize.py  
|   |-- standardize.py  
|-- regression  
|   |-- __init__.py  
|   |-- regression.py  
|-- utils.py
```

Importing modules

We imported `preprocessing` into `mysklearn`

```
import mysklearn
help(mysklearn.preprocessing)
```

```
Help on package mysklearn.preprocessing in
mysklearn:
```

```
NAME
mysklearn.preprocessing - A subpackage
for standard preprocessing operations.
```

But `preprocessing` has no link to `normalize`

```
import mysklearn
help(mysklearn.preprocessing.normalize)
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: module
'mysklearn.preprocessing' has no attribute
'normalize'
```

Importing modules

`mysklearn/preprocessing/__init__.py`

Absolute import

```
from mysklearn.preprocessing import normalize
```

Relative import

```
from . import normalize
```

Directory tree for package with subpackages

```
mysklearn/  
|-- __init__.py  
|-- preprocessing  
|   |-- __init__.py    <--  
|   |-- normalize.py  
|   |-- standardize.py  
|-- regression  
|   |-- __init__.py  
|   |-- regression.py  
|-- utils.py
```

Restructuring imports

```
import mysklearn
```

```
help(mysklearn.preprocessing.normalize.normalize_data)
```

```
Help on function normalize_data in module  
mysklearn.preprocessing.normalize:
```

```
normalize_data(x)  
    Normalize the data array.
```

Import function into subpackage

`mysklearn/preprocessing/__init__.py`

Absolute import

```
from mysklearn.preprocessing.normalize import \
    normalize_data
```

Relative import

```
from .normalize import normalize_data
```

Directory tree for package with subpackages

```
mysklearn/
|-- __init__.py
|-- preprocessing
|   |-- __init__.py    <--
|   |-- normalize.py
|   |-- standardize.py
|-- regression
|   |-- __init__.py
|   |-- regression.py
|-- utils.py
```

Import function into subpackage

```
import mysklearn
```

```
help(mysklearn.preprocessing.normalize_data)
```

```
Help on function normalize_data in module  
mysklearn_imp.preprocessing.normalize:
```

```
normalize_data(x)  
    Normalize the data array.
```


Importing between sibling modules

In `normalize.py`

Absolute import

```
from mysklearn.preprocessing.funcs import (  
    mymax, mymin  
)
```

Relative import

```
from .funcs import mymax, mymin
```

Directory tree for package with subpackages

```
mysklearn/  
|-- __init__.py  
|-- preprocessing  
|   |-- __init__.py  
|   |-- normalize.py <--  
|   |-- funcs.py  
|   |-- standardize.py  
|-- regression  
|   |-- __init__.py  
|   |-- regression.py  
|-- utils.py
```

Importing between modules far apart

A custom exception `MyException` is in `utils.py`

In `normalize.py`, `standardize.py` and `regression.py`

Absolute import

```
from mysklearn.utils import MyException
```

Relative import

```
from ..utils import MyException
```

Directory tree for package with subpackages

```
mysklearn/  
|-- __init__.py  
|-- preprocessing  
|   |-- __init__.py  
|   |-- normalize.py <--  
|   `-- standardize.py <--  
|-- regression  
|   |-- __init__.py  
|   |-- regression.py <--  
`-- utils.py
```

Relative import cheat sheet

- `from . import module`
 - From current directory, import `module`
- `from .. import module`
 - From one directory up, import `module`
- `from .module import function`
 - From module in current directory, import `function`
- `from ..subpackage.module import function`
 - From subpackage one directory up, from `module` in that subpackage, import `function`

Let's practice!

DEVELOPING PYTHON PACKAGES