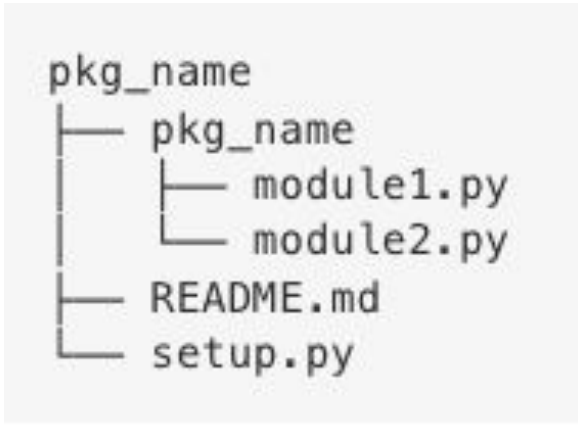
Lesson 3b: Creating Packages with Python

Reference:
Research Software Engineering with Python by Damien Irving, Kate Hertweck, Luke Johnston, Joel Ostblom, Charlotte Wickham, and Greg Wilson
https://merely-useful.tech/py-rse/packaging.html

Creating a Python Package

A generic package folder hierarchy has:

- a top level directory names after the package
- that contains a directory that is also named after the package
- that contains the package's source files



```
pkg_name
├── pkg_name
│   ├── module1.py
│   └── module2.py
├── README.md
└── setup.py
```

Search the Python Package Index https://pypi.org/ to check which names are already taken!

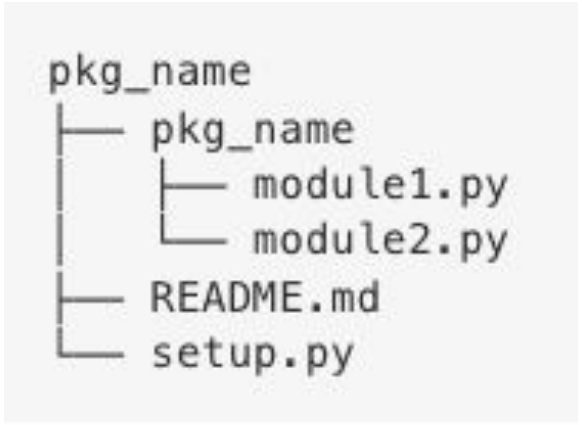
Creating a Python package

Using setuptools allows everyone, regardless of Python distribution, to use our package.

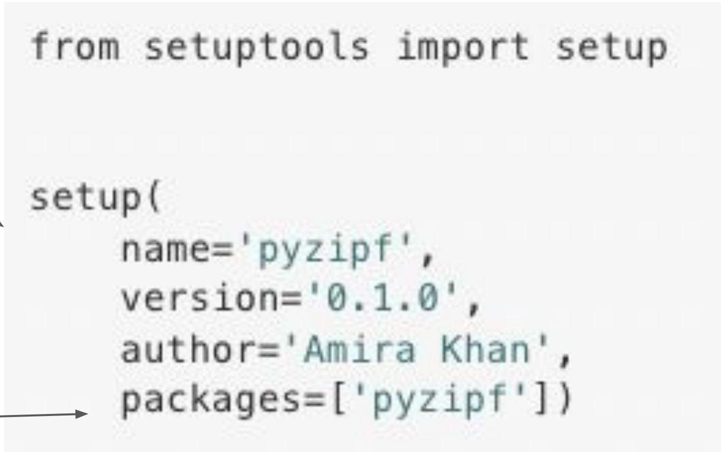
To use setup, we have to make a file called setup.py in the directory above the root directory.

The set up file looks like this:

(Packages parameter is straightforward, since we only have one package directory. In more complex projects, the find_packages function from setuptools can automatically find all packages)



```
pkg_name
├── pkg_name
│   ├── module1.py
│   └── module2.py
├── README.md
└── setup.py
```



```
from setuptools import setup


setup(
    name='pyzipf',
    version='0.1.0',
    author='Amira Khan',
    packages=['pyzipf'])
```

Virtual Environments

Virtual environments isolate package dependencies from the main Python installation.

They:

- Allow easy install/uninstall for testing
- Ensure package works in an empty environment
- Are created with conda or virtualenv

Creating a virtual environment:

```
$ conda create -n pyzipf pip python=3.7.6
```

Activating it:

```
$ conda activate pyzipf
```

(`$ conda deactivate` to deactivate)

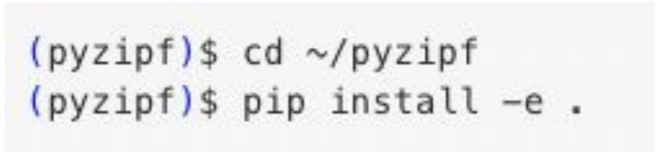
Once we've done this, the python command runs the interpreter in pyzipf/bin:

```
(pyzipf)$ which python
```

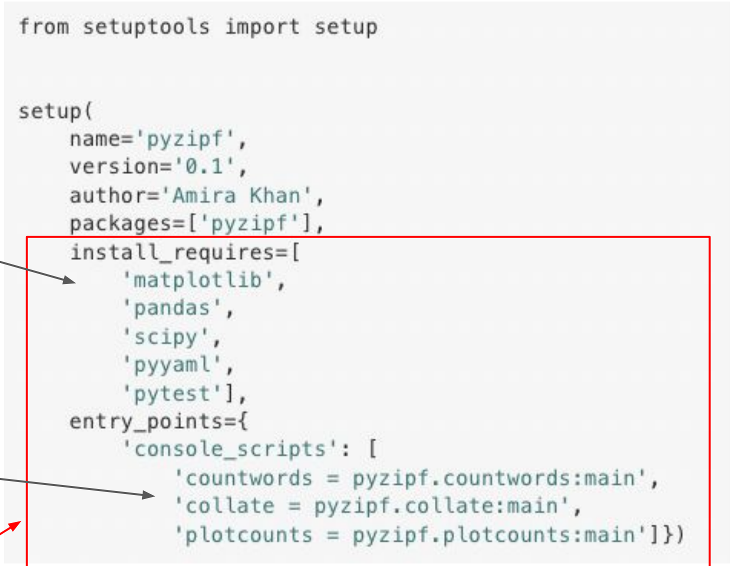
```
/Users/amira/anaconda3/envs/pyzipf/bin/python
```

Installing a Development Package

- -e indicates that we want to install the package in "editable" mode
  - which means that any changes we make in the package code are directly available to use without having to reinstall the package
- The . means "install from the current directory."
- List dependencies with install_requires in setup.py
  - Lets pip handle dependencies automatically
- Make scripts executable as command line tools via entry_points
  - Map script functions to command names



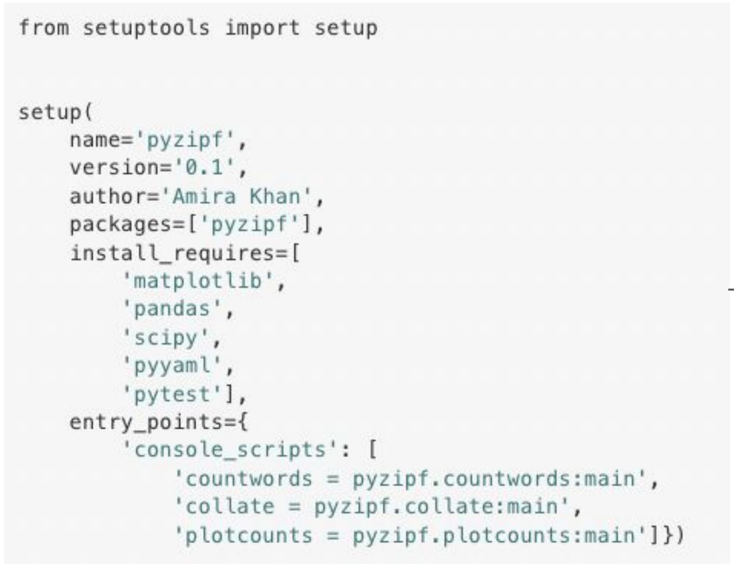
```
(pyzipf)$ cd ~/pyzipf
(pyzipf)$ pip install -e .
```



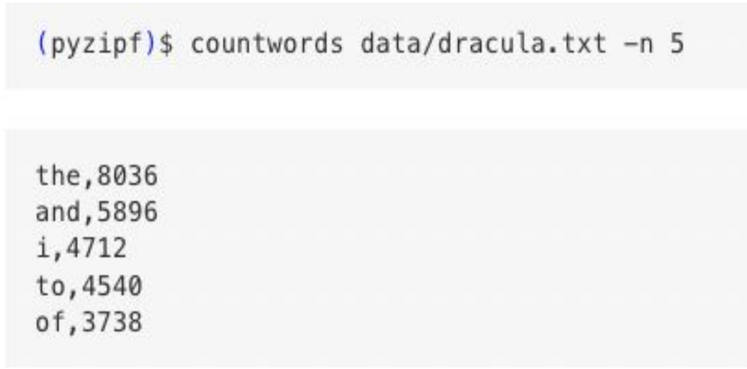
```
from setuptools import setup


setup(
    name='pyzipf',
    version='0.1',
    author='Amira Khan',
    packages=['pyzipf'],
    install_requires=[
        'matplotlib',
        'pandas',
        'scipy',
        'pyyaml',
        'pytest'],
    entry_points={
        'console_scripts': [
            'countwords = pyzipf.countwords:main',
            'collate = pyzipf.collate:main',
            'plotcounts = pyzipf.plotcounts:main']})
```

Add to setup.py

… and now we can use commands directly from the Unix shell!



```python
from setuptools import setup


setup(
    name='pyzipf',
    version='0.1',
    author='Amira Khan',
    packages=['pyzipf'],
    install_requires=[
        'matplotlib',
        'pandas',
        'scipy',
        'pyyaml',
        'pytest'],
    entry_points={
        'console_scripts': [
            'countwords = pyzipf.countwords:main',
            'collate = pyzipf.collate:main',
            'plotcounts = pyzipf.plotcounts:main']})
```



```
(pyzipf)$ countwords data/dracula.txt -n 5
```

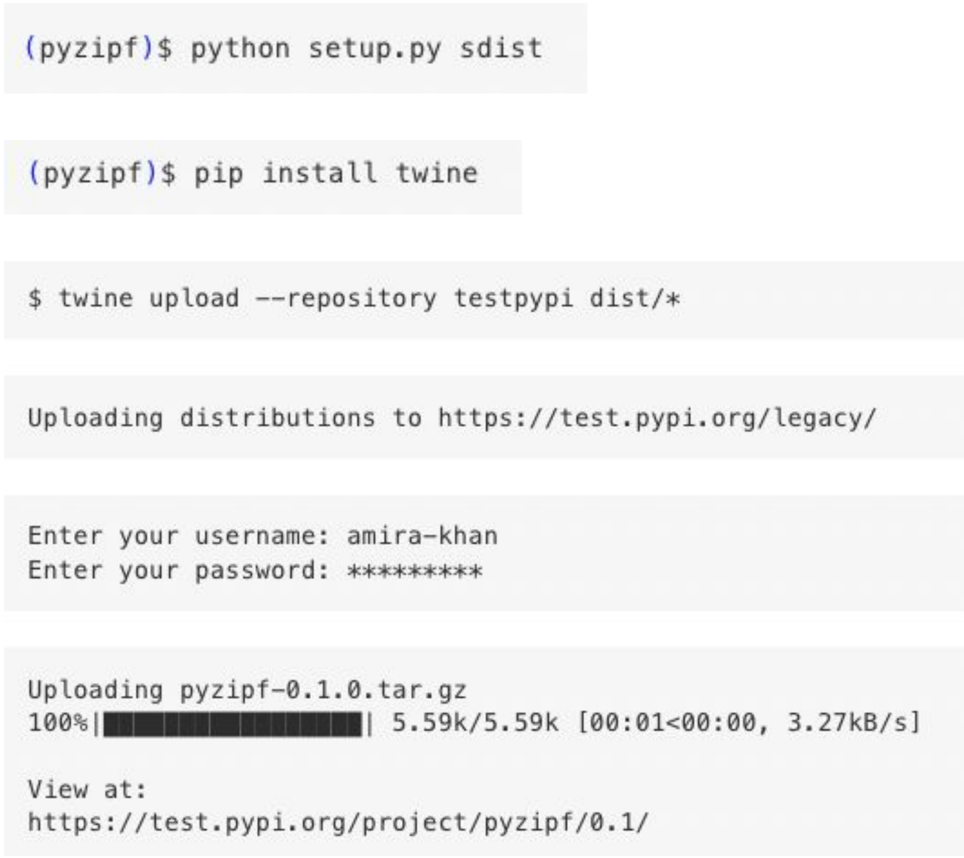
```
the,8036
and,5896
i,4712
to,4540
of,3738
```

Distributing Packages

In order for people to run pip install pyzipf to use out package, we need to use setuptools to create a **source distribution**. Then:

1) Make a free account on PyPI
2) Install twine, the preferred tool for uploading packages to PyPI
3) Follow the Python Packaging User guide to upload our distribution from the dist folder, using the --repository option to specify the TestPyPI repo

… and we're happy with our package at TestPyPI, we can go through the same process to put it on the main PyPI repo.



```
(pyzipf)$ python setup.py sdist
```

```
(pyzipf)$ pip install twine
```

```
$ twine upload --repository testpypi dist/*
```

```
Uploading distributions to https://test.pypi.org/legacy/
```

```
Enter your username: amira-khan
Enter your password: *********
```

```
Uploading pyzipf-0.1.0.tar.gz
100%|██████████████████| 5.59k/5.59k [00:01<00:00, 3.27kB/s]

View at:
https://test.pypi.org/project/pyzipf/0.1/
```

Documenting Packages

Sphinx:

- a document generator for more complex Python packages, often used in combination with a free online hosting service called Read the Docs
- can scan Python code for function names and docstrings and export that information to HTML format for hosting on the web
- uses a format called reStructuredText (reST)
  - plain-text markup format that can be rendered to HTML or PDF

Renaming our README to an rst:

- note that there are formatting differences for rst files

```
$ git mv README.md README.rst
```

Documenting Packages

Install Sphinx and create a docs/ directory at the top of the repo

```
$ pip install sphinx
$ mkdir docs
$ cd docs
```

Use quickstart to create a minimal set of documentation that includes the package-level info in the README.rst, and function-level info in the docstrings made along the way

```
$ sphinx-quickstart
```

Documenting Packages

Quickstart creates a file called `conf.py` in the docs directory that configures Sphinx. First, we need to add the following code to the Path setup:

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

We will also change the “general configuration” section to add autodoc to the list of Sphinx extensions we want:

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

Documenting Packages

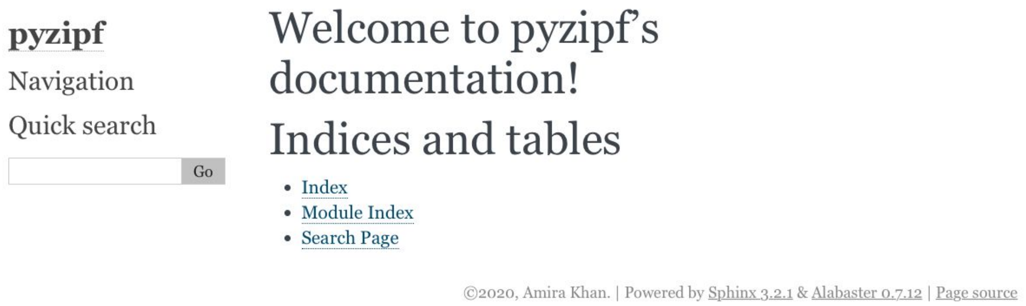
We can now generate a Sphinx `autodoc` script that generates information about each of our modules and puts it in corresponding `.rst` files in the `docs/source` directory:

```
sphinx-apidoc -o source/ ../pyzipf
```

```
Creating file source/collate.rst.
Creating file source/countwords.rst.
Creating file source/plotcounts.rst.
Creating file source/test_zipfs.rst.
Creating file source/utilities.rst.
Creating file source/modules.rst.
```

… and now we can generate our webpage!

If we run `make html` and open `docs/_build/index.html` in a web browser, we'll see a landing page.


pyzipf

Navigation

Quick search

Go

Welcome to pyzipf's documentation!

Indices and tables

- Index
- Module Index
- Search Page

©2020, Amira Khan. | Powered by Sphinx 3.2.1 & Alabaster 0.7.12 | Page source

Now, we create a `requirements_docs.txt` file that contains this line (where the version number is found by running `pip freeze`):

```
Sphinx>=1.7.4
```

Anyone wanting to build the documentation (including us, on another computer) now only needs run `pip install -r requirements_docs.txt`

Hosting Documentation Online

Read the Docs

- a community-supported site that hosts software documentation free of charge.
- integrates with GitHub so that documentation is automatically re-built every time updates are pushed to the project's GitHub repository

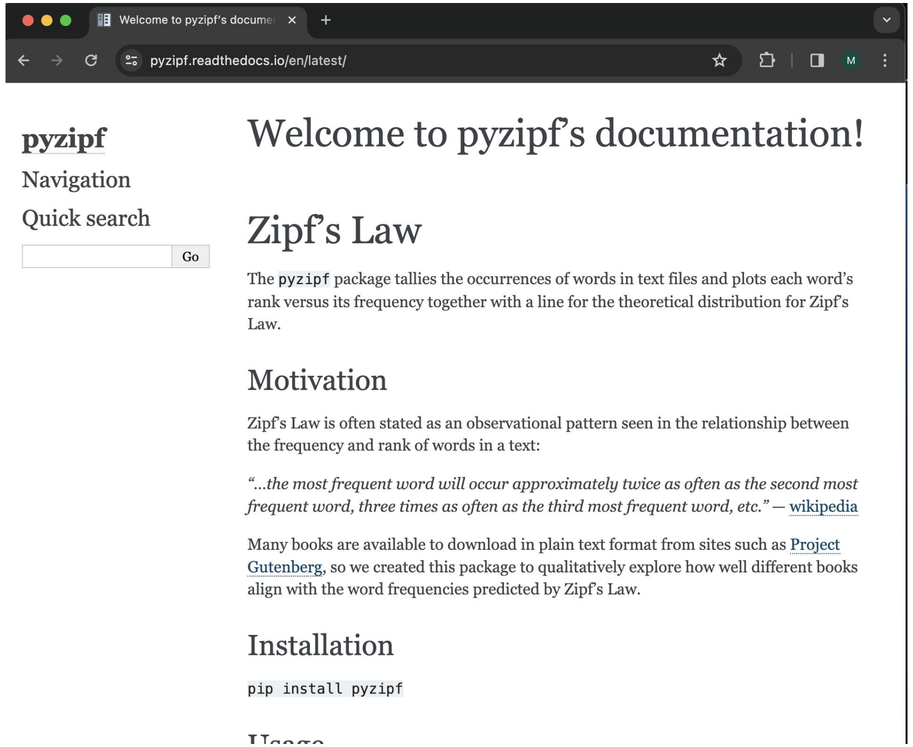
We need to create and save a Read the Docs configuration file in the root directory of our `pyzipf` package:

```
$ cd ~/pyzipf
$ cat .readthedocs.yml
```

```
# .readthedocs.yml
# Read the Docs configuration file
# See https://docs.readthedocs.io/en/stable/config-file/v2.html
# for details

# Required
version: 2

# Build documentation in the docs/ directory with Sphinx
sphinx:
  configuration: docs/conf.py

# Optionally set the version of Python and requirements required
# to build your docs
python:
  version: 3.7
  install:
    - requirements: requirements.txt
```

Hosting Documentation Online

https://pyzipf.readthedocs.io/



Welcome to pyzipf's documentation!

Zipf's Law

The `pyzipf` package tallies the occurrences of words in text files and plots each word's rank versus its frequency together with a line for the theoretical distribution for Zipf's Law.

Motivation

Zipf's Law is often stated as an observational pattern seen in the relationship between the frequency and rank of words in a text:

"...the most frequent word will occur approximately twice as often as the second most frequent word, three times as often as the third most frequent word, etc." — wikipedia

Many books are available to download in plain text format from sites such as Project Gutenberg, so we created this package to qualitatively explore how well different books align with the word frequencies predicted by Zipf's Law.

Installation

```
pip install pyzipf
```

Software Journals

- Zenodo integrates with GitHub so that we can obtain a DOI
- Some research disciplines have journals devoted to describing particular types of software (e.g., Geoscientific Model Development)
- There are also generic software journals such as the Journal of Open Research Software and the Journal of Open Source Software.
- Once you've obtained a DOI and possibly published with a software journal, the last step is to tell users how to cite your new software package

```
$ cat CITATION.md
```

```
# Citation

If you use the pyzipf package for work/research presented in a
publication, we ask that you please cite:

Khan A and Virtanen S, 2020. pyzipf: A Python package for word
count analysis. *Journal of Important Software*, 5(51), 2317,
https://doi.org/10.21105/jois.02317

### BibTeX entry

@article{Khan2020,
    title={pyzipf: A Python package for word count analysis.},
    author={Khan, Amira and Virtanen, Sami},
    journal={Journal of Important Software},
    volume={5},
    number={51},
    eid={2317},
    year={2020},
    doi={10.21105/jois.02317},
    url={https://doi.org/10.21105/jois.02317},
}
```