Anatomy and Evolution of a Python project

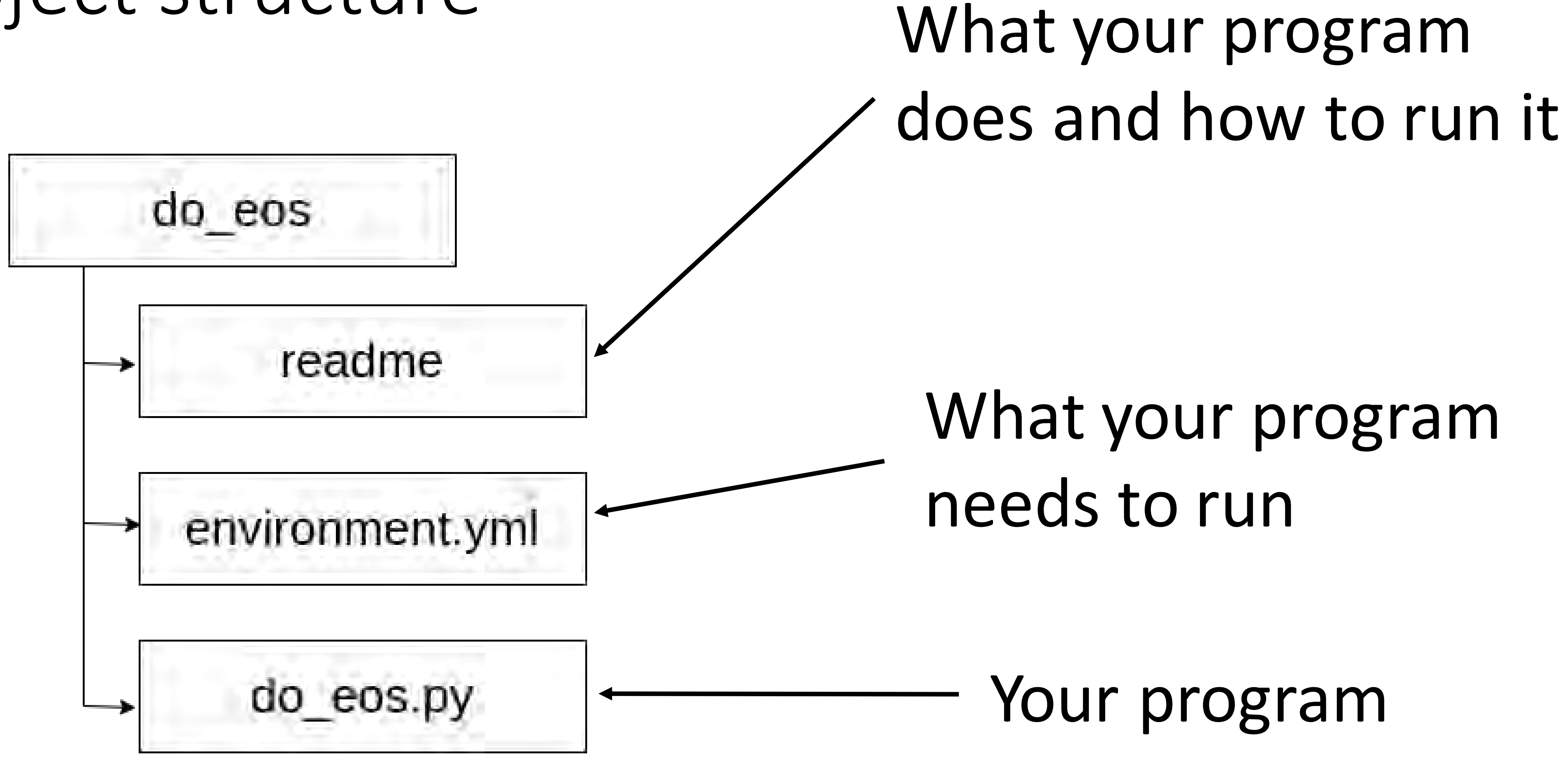
What CodeAcademy does not teach you

First Of All

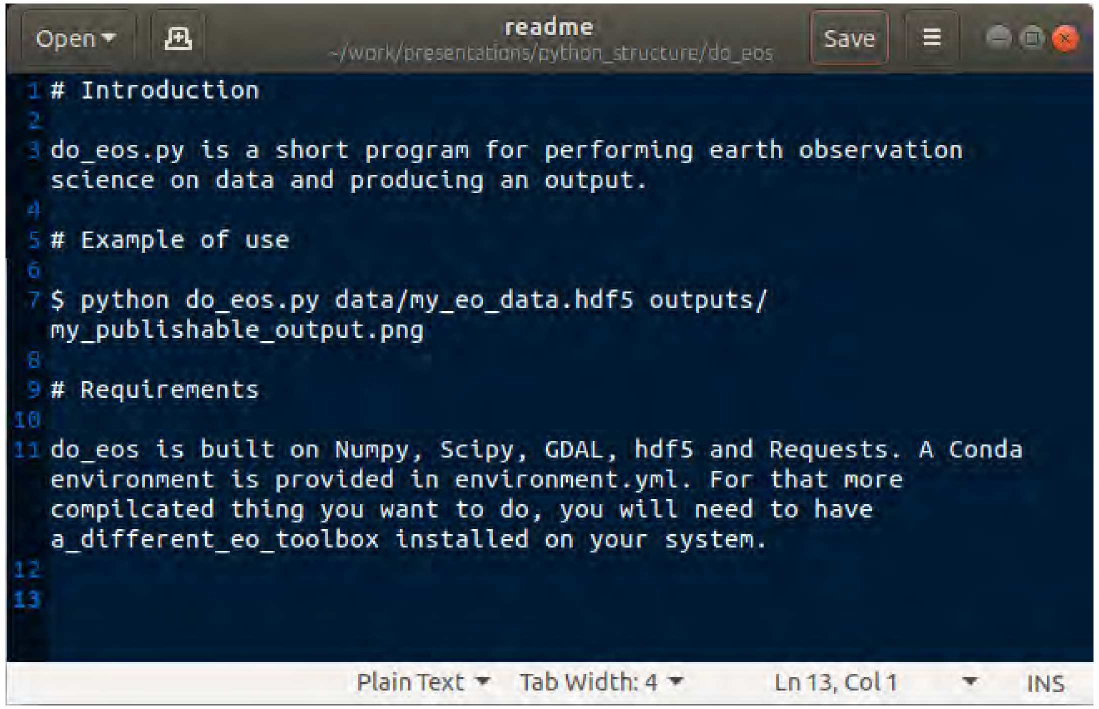
You Are Writing Code To Be Read By Humans

The Anatomy Of A Python Project

Project structure


do_eos
readme
environment.yml
do_eos.py
What your program does and how to run it
What your program needs to run
Your program

Readme

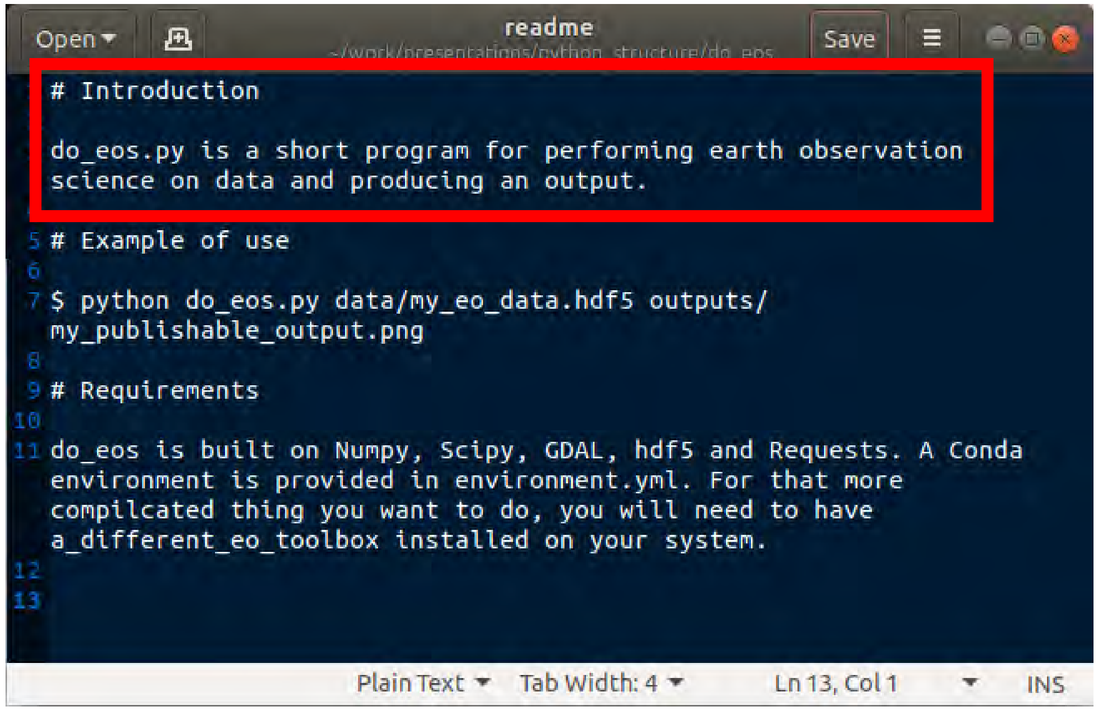


```
# Introduction

do_eos.py is a short program for performing earth observation
science on data and producing an output.

# Example of use

$ python do_eos.py data/my_eo_data.hdf5 outputs/
my_publishable_output.png

# Requirements

do_eos is built on Numpy, Scipy, GDAL, hdf5 and Requests. A Conda
environment is provided in environment.yml. For that more
compilcated thing you want to do, you will need to have
a_different_eo_toolbox installed on your system.
```

A file readable in plain text

Tells the user at least the minimum they need to get started and run the program

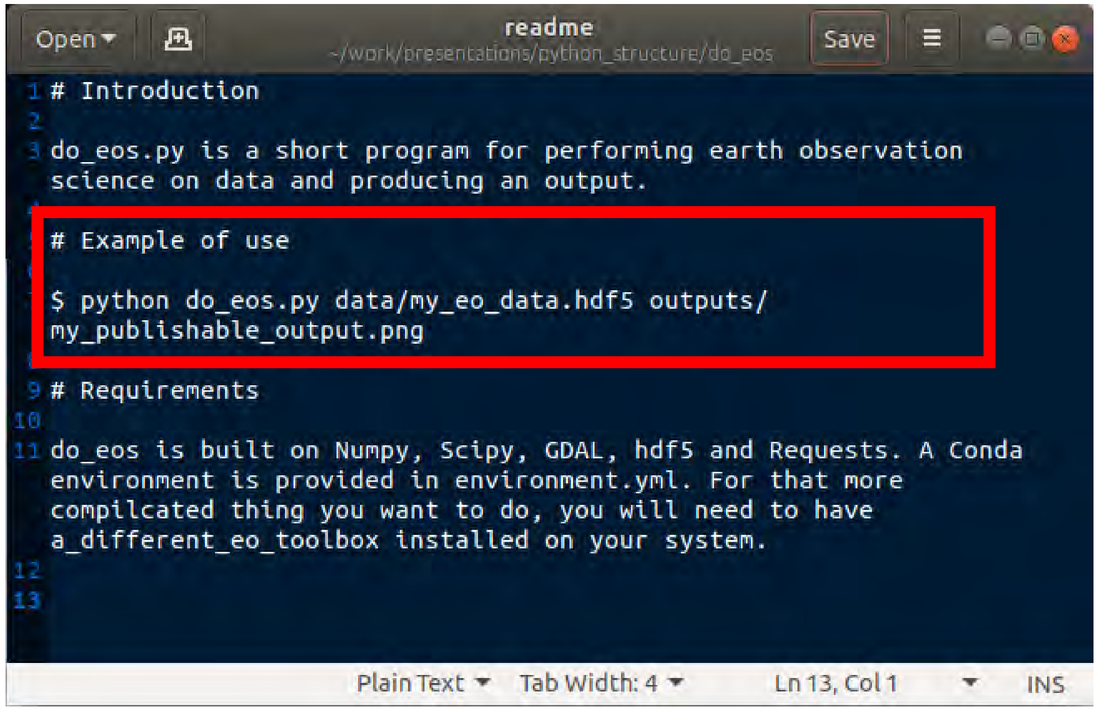
Readme: introduction



```
# Introduction

do_eos.py is a short program for performing earth observation
science on data and producing an output.

# Example of use

$ python do_eos.py data/my_eo_data.hdf5 outputs/
my_publishable_output.png

# Requirements

do_eos is built on Numpy, Scipy, GDAL, hdf5 and Requests. A Conda
environment is provided in environment.yml. For that more
compilcated thing you want to do, you will need to have
a_different_eo_toolbox installed on your system.
```

- What your program does and why

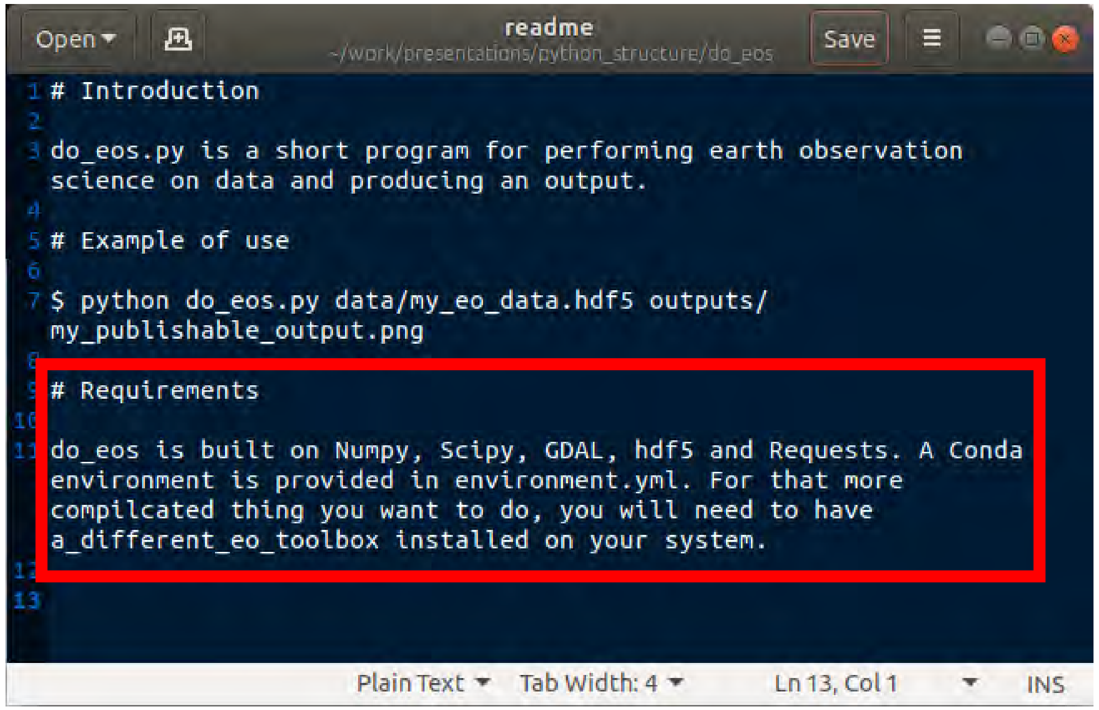
Readme: Example of use



```
# Introduction

do_eos.py is a short program for performing earth observation
science on data and producing an output.

# Example of use

$ python do_eos.py data/my_eo_data.hdf5 outputs/
my_publishable_output.png

# Requirements

do_eos is built on Numpy, Scipy, GDAL, hdf5 and Requests. A Conda
environment is provided in environment.yml. For that more
compilcated thing you want to do, you will need to have
a_different_eo_toolbox installed on your system.
```

- How to call your program
- This is what people are here for: they will try this first

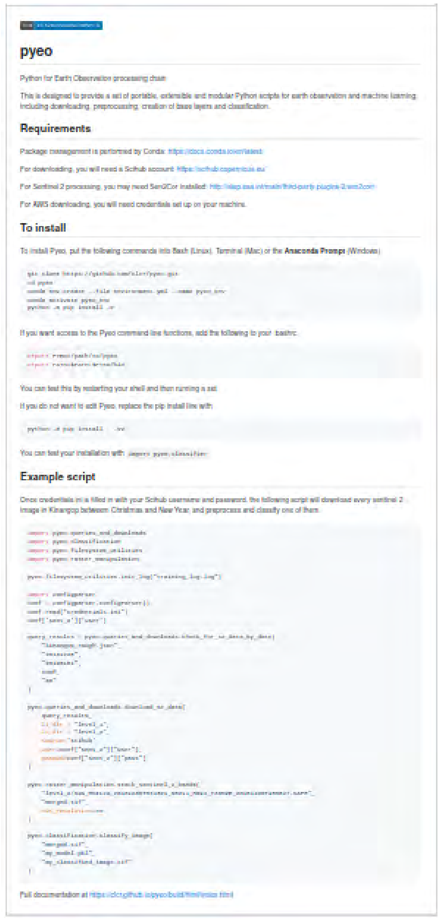
Readme



```
# Introduction

do_eos.py is a short program for performing earth observation
science on data and producing an output.

# Example of use

$ python do_eos.py data/my_eo_data.hdf5 outputs/
my_publishable_output.png

# Requirements

do_eos is built on Numpy, Scipy, GDAL, hdf5 and Requests. A Conda
environment is provided in environment.yml. For that more
compilcated thing you want to do, you will need to have
a_different_eo_toolbox installed on your system.
```

- An overview of what your program needs to run
- Possibly how to install it

Advanced Readme



DOI [illegible]

pyeo

Python for Earth Observation processing chain

This is designed to provide a set of portable, extensible and modular Python scripts for earth observation and machine learning, including downloading, preprocessing, creation of base layers and classification.

Requirements

Package management is performed by Conda: https://docs.conda.io/en/latest/

For downloading, you will need a Scihub account: https://scihub.copernicus.eu/

For Sentinel 2 processing, you may need Sen2Cor installed: http://step.esa.int/main/third-party-plugins-2/sen2cor/

For AWS downloading, you will need credentials set up on your machine.

To install

To install Pyeo, put the following commands into Bash (Linux), Terminal (Mac) or the **Anaconda Prompt** (Windows)

```
git clone https://github.com/clcr/pyeo.git
cd pyeo
conda env create --file environment.yml --name pyeo_env
conda activate pyeo_env
python -m pip install -e .
```

If you want access to the Pyeo command line functions, add the following to your .bashrc

```
export PYEO=/path/to/pyeo
export PATH=$PATH:$PYEO/bin
```

You can test this by restarting your shell and then running [illegible]

If you do not want to edit Pyeo, replace the pip install line with

```
python -m pip install . -vv
```

You can test your installation with `import pyeo.classification`

Example script

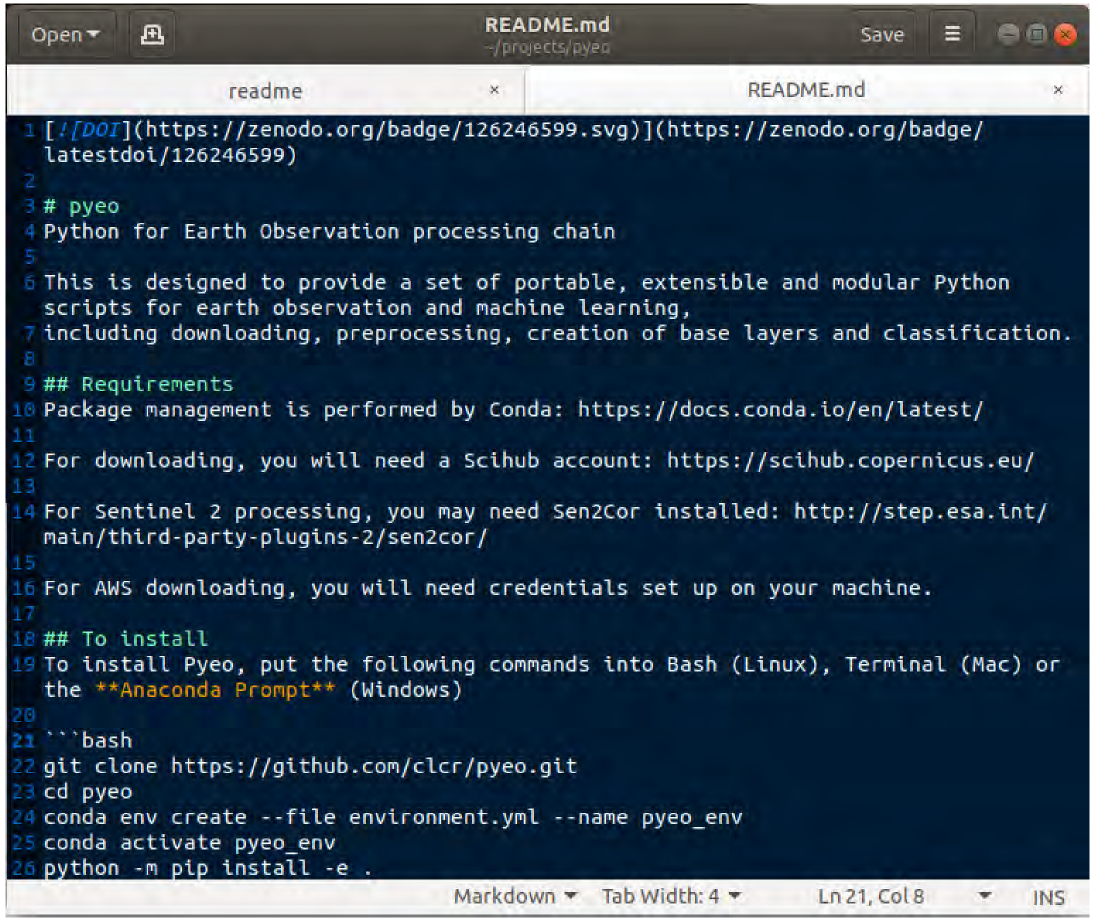
Once credentials.ini is filled in with your Scihub username and password, the following script will download every sentinel 2 image in Kinangop between Christmas and New Year, and preprocess and classify one of them.

```
import pyeo.queries_and_downloads
import pyeo.classification
import pyeo.filesystem_utilities
import pyeo.raster_manipulation

pyeo.filesystem_utilities.init_log("training_log.log")

import configparser
conf = configparser.ConfigParser()
conf.read("credentials.ini")
conf["sent_2"]["user"]

query_results = pyeo.queries_and_downloads.check_for_s2_data_by_date(
    "kinangop_rough.json",
    "20181224",
    "20190105",
    conf,
    "50"
)

pyeo.queries_and_downloads.download_s2_data(
    query_results,
    l1_dir = "level_1",
    l2_dir = "level_2",
    source='scihub',
    user=conf["sent_2"]["user"],
    passwd=conf["sent_2"]["pass"]
)

pyeo.raster_manipulation.stack_sentinel_2_bands(
    "level_2/[illegible].SAFE",
    "merged.tif",
    out_resolution=10
)

pyeo.classification.classify_image(
    "merged.tif",
    "my_model.pkl",
    "my_classified_image.tif"
)
```

Full documentation at https://clcr.github.io/pyeo/build/html/index.html

- Can do many things with Readmes and github or others:
  - Nice formatting
  - Embedded code examples
  - Links
  - Citable DOIs

Advanced Readme



~~~
[![DOI](https://zenodo.org/badge/126246599.svg)](https://zenodo.org/badge/latestdoi/126246599)

# pyeo
Python for Earth Observation processing chain

This is designed to provide a set of portable, extensible and modular Python scripts for earth observation and machine learning,
including downloading, preprocessing, creation of base layers and classification.

## Requirements
Package management is performed by Conda: https://docs.conda.io/en/latest/

For downloading, you will need a Scihub account: https://scihub.copernicus.eu/

For Sentinel 2 processing, you may need Sen2Cor installed: http://step.esa.int/main/third-party-plugins-2/sen2cor/

For AWS downloading, you will need credentials set up on your machine.

## To install
To install Pyeo, put the following commands into Bash (Linux), Terminal (Mac) or the **Anaconda Prompt** (Windows)

```bash
git clone https://github.com/clcr/pyeo.git
cd pyeo
conda env create --file environment.yml --name pyeo_env
conda activate pyeo_env
python -m pip install -e .
~~~

- Can do many things with Readmes and github or others:
  - Nice formatting
  - Embedded code examples
  - Links
  - Citable DOIs
- But keep it readable in standard text
  - This one made in Markdown
  - Do not worry about this for now.

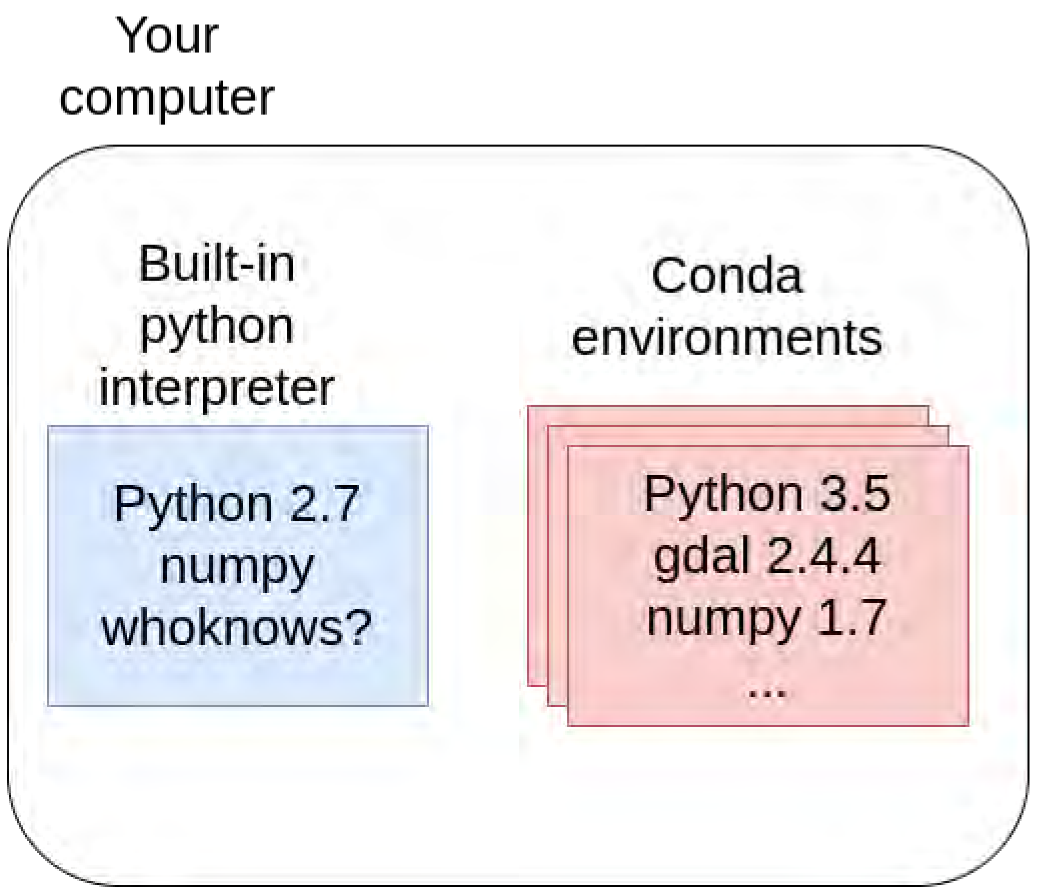
Environment.yml and package management

R has CRAN

Python has Conda

And pip

Python, Conda and environments

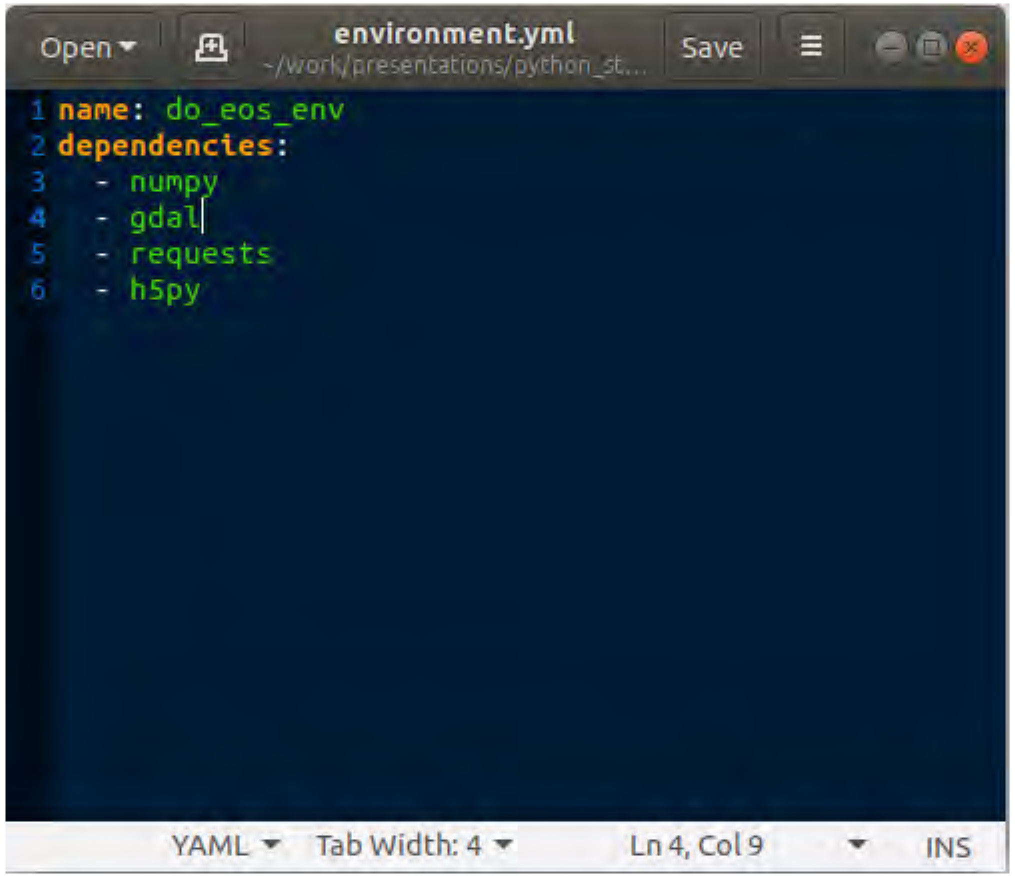


Provides special Python environments that can be turned on and off

Different libraries and versions of Python in each one

Allows **isolation** between Python programs

Environment.yml



```yaml
name: do_eos_env
dependencies:
    - numpy
    - gdal
    - requests
    - h5py
```

A list of packages your program needs to run

Can **create** an environment with

$ conda env create -f environment.yml

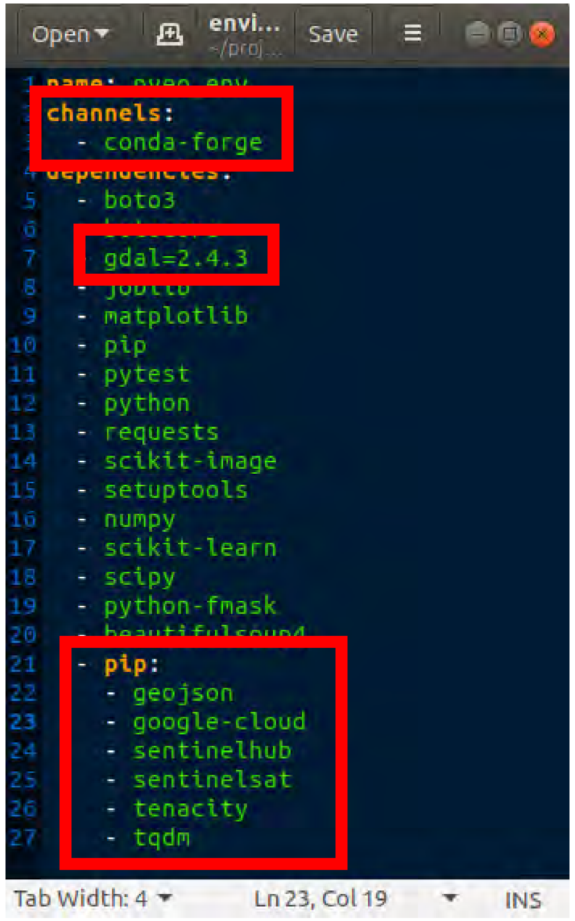
Can **update** an environment with

$ conda env update -f environment.yml

Can **use** an environment with

$ conda activate do_eos_env

Advanced environment.yml



```yaml
name: pyeo_env
channels:
  - conda-forge
dependencies:
  - boto3
  - botocore
  - gdal=2.4.3
  - joblib
  - matplotlib
  - pip
  - pytest
  - python
  - requests
  - scikit-image
  - setuptools
  - numpy
  - scikit-learn
  - scipy
  - python-fmask
  - beautifulsoup4
  - pip:
    - geojson
    - google-cloud
    - sentinelhub
    - sentinelsat
    - tenacity
    - tqdm
```

- Can specify **conda channels**
  - Different sources for libraries
- Can **pin versions** of libraries
- Can specify installs from **pip**
  - Pip: more packages, less quality control

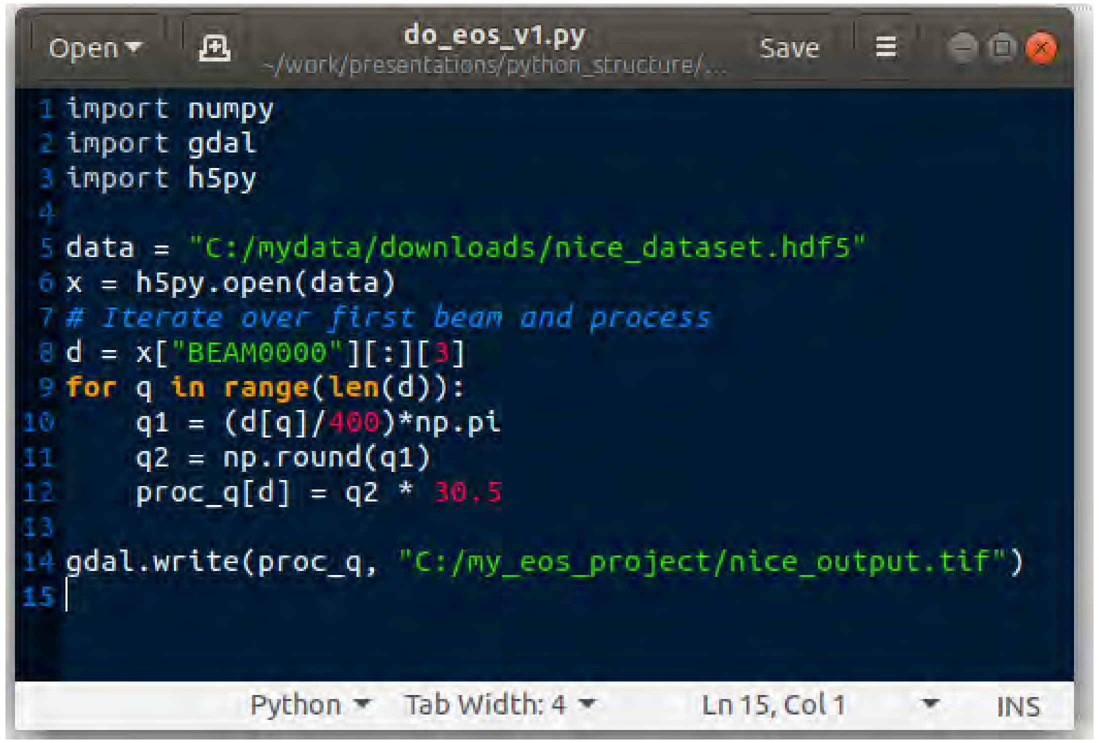
IMPORTANT

THE MORE EXTERNAL LIBRARIES YOU USE

THE MORE LIKELY YOUR PROGRAM IS TO BREAK IN THE FUTURE

The Evolution Of A Python Project

do_eos.py: version 1: "It worked!"



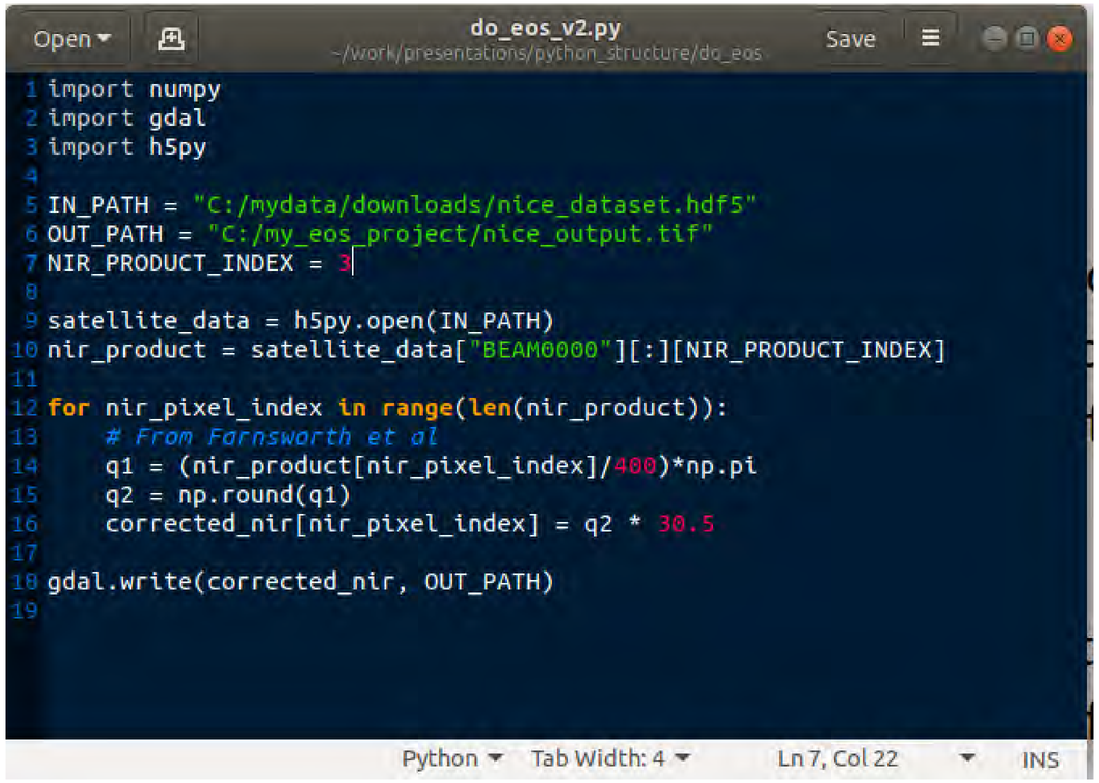
```python
import numpy
import gdal
import h5py

data = "C:/mydata/downloads/nice_dataset.hdf5"
x = h5py.open(data)
# Iterate over first beam and process
d = x["BEAM0000"][:][3]
for q in range(len(d)):
    q1 = (d[q]/400)*np.pi
    q2 = np.round(q1)
    proc_q[d] = q2 * 30.5

gdal.write(proc_q, "C:/my_eos_project/nice_output.tif")
```

- The computer understands it
- Fine for you *right now*

BUT:

- Written in a hurry
- Proof of concept
- Not intuitive for you in the future

do_eos.py: version 2: "Can I use your script?"



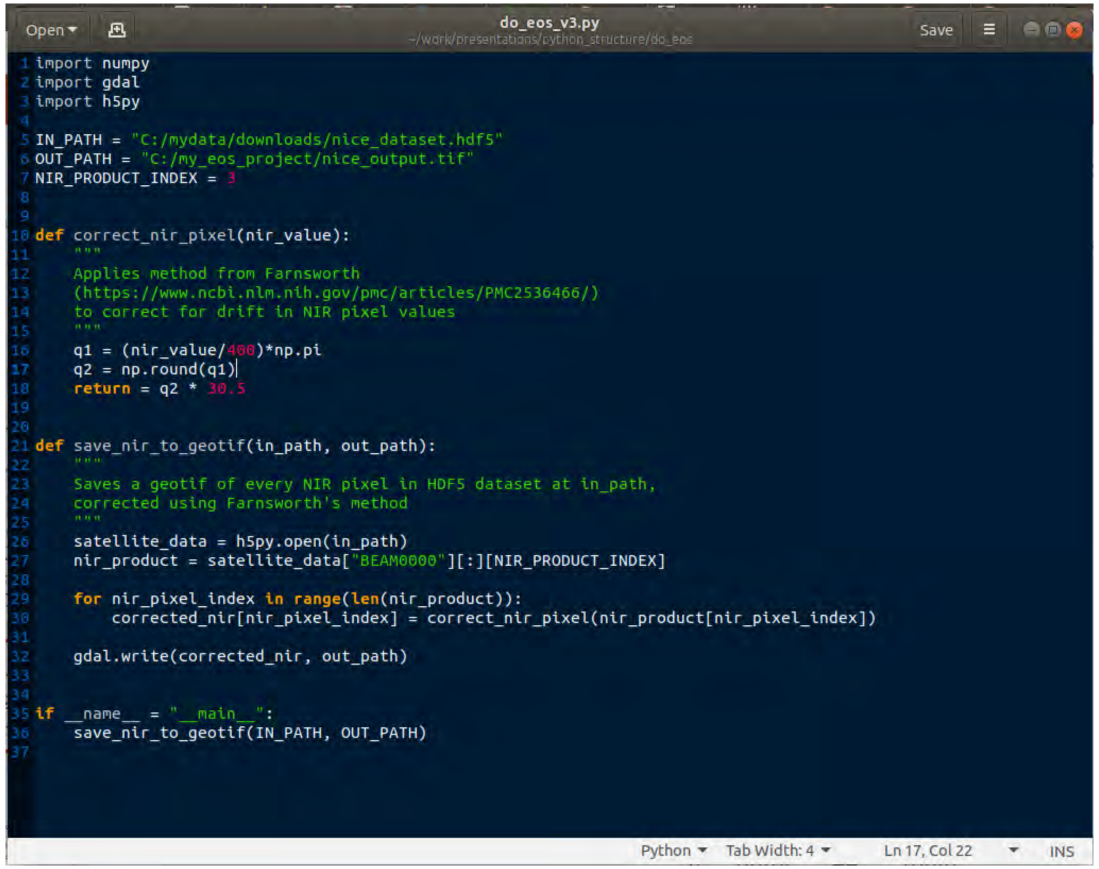
```python
import numpy
import gdal
import h5py

IN_PATH = "C:/mydata/downloads/nice_dataset.hdf5"
OUT_PATH = "C:/my_eos_project/nice_output.tif"
NIR_PRODUCT_INDEX = 3

satellite_data = h5py.open(IN_PATH)
nir_product = satellite_data["BEAM0000"][:][NIR_PRODUCT_INDEX]

for nir_pixel_index in range(len(nir_product)):
    # From Farnsworth et al
    q1 = (nir_product[nir_pixel_index]/400)*np.pi
    q2 = np.round(q1)
    corrected_nir[nir_pixel_index] = q2 * 30.5

gdal.write(corrected_nir, OUT_PATH)
```

- Made when someone else wants to process the data
- Paths and other 'changeables' moved to top of program
- Variables renamed
  - 'find and replace' (text editors)
  - 'Refactor -> rename' (IDEs)

•Can now be used for different data more easily

•But: Interior of loop still a mystery

do_eos.py: version 3: "I'd like to put it in my processing chain"



```python
import numpy
import gdal
import h5py

IN_PATH = "C:/mydata/downloads/nice_dataset.hdf5"
OUT_PATH = "C:/my_eos_project/nice_output.tif"
NIR_PRODUCT_INDEX = 3


def correct_nir_pixel(nir_value):
    """
    Applies method from Farnsworth
    (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2536466/)
    to correct for drift in NIR pixel values
    """
    q1 = (nir_value/400)*np.pi
    q2 = np.round(q1)
    return = q2 * 30.5


def save_nir_to_geotif(in_path, out_path):
    """
    Saves a geotif of every NIR pixel in HDF5 dataset at in_path,
    corrected using Farnsworth's method
    """
    satellite_data = h5py.open(in_path)
    nir_product = satellite_data["BEAM0000"][:][NIR_PRODUCT_INDEX]

    for nir_pixel_index in range(len(nir_product)):
        corrected_nir[nir_pixel_index] = correct_nir_pixel(nir_product[nir_pixel_index])

    gdal.write(corrected_nir, out_path)


if __name__ = "__main__":
    save_nir_to_geotif(IN_PATH, OUT_PATH)

```

- Made when someone wants to expand on the method
- Program broken into **functions**
  - Functions have **docstrings**
  - Functions can be **imported** by othe programs
- Program entry point is explicit

  if __name__ == "__main__":

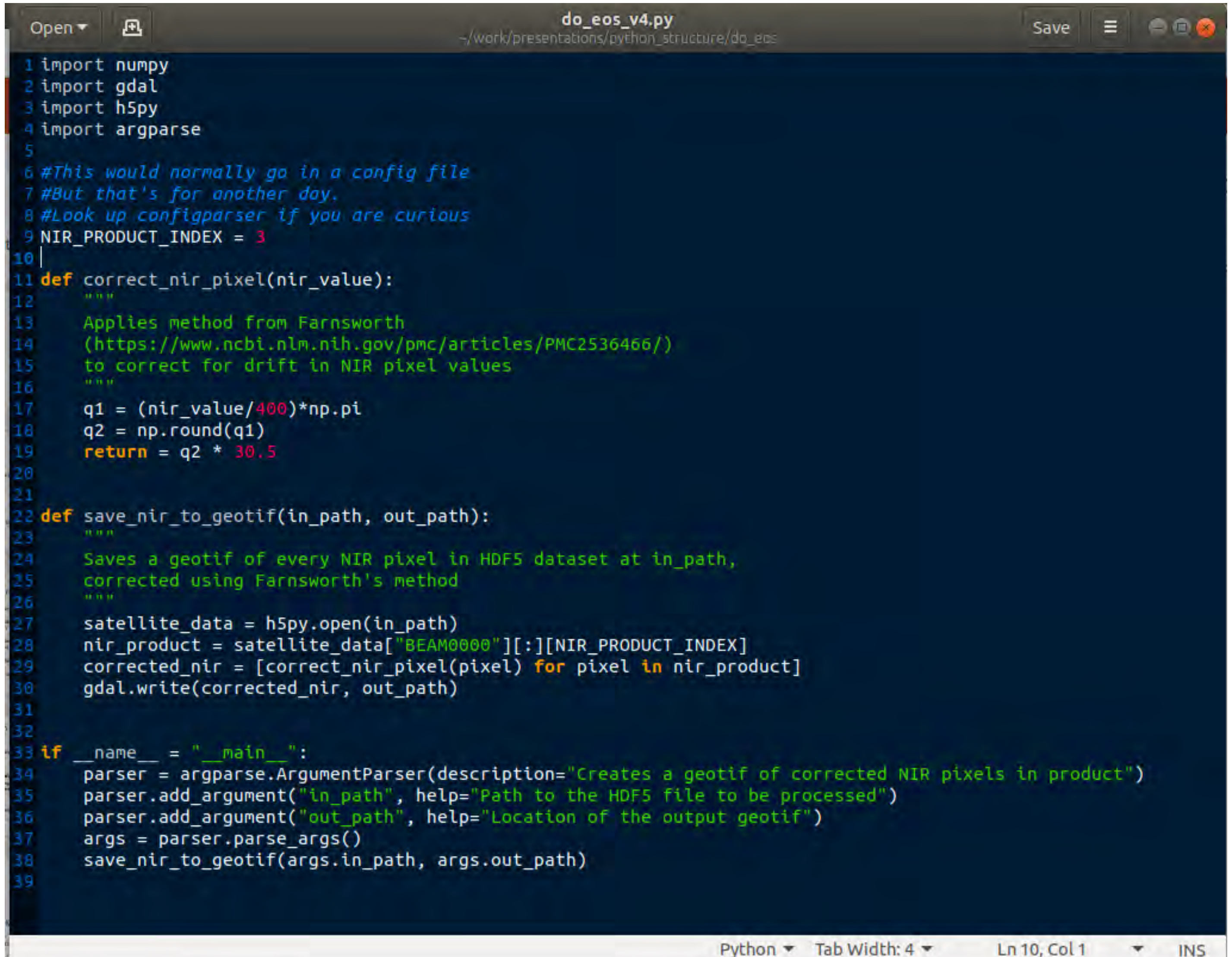
The Tau Of Functions

A Function is a **Single Idea**

A Function should perform **One Task**

"functions should be strung out like pearls on a necklace"
https://www.mit.edu/~xela/tao.html (sort of)

do_eos.py: version 4: "Can you run it on ALICE?"



```python
import numpy
import gdal
import h5py
import argparse

#This would normally go in a config file
#But that's for another day.
#Look up configparser if you are curious
NIR_PRODUCT_INDEX = 3

def correct_nir_pixel(nir_value):
    """
    Applies method from Farnsworth
    (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2536466/)
    to correct for drift in NIR pixel values
    """
    q1 = (nir_value/400)*np.pi
    q2 = np.round(q1)
    return = q2 * 30.5


def save_nir_to_geotif(in_path, out_path):
    """
    Saves a geotif of every NIR pixel in HDF5 dataset at in_path,
    corrected using Farnsworth's method
    """
    satellite_data = h5py.open(in_path)
    nir_product = satellite_data["BEAM0000"][:][NIR_PRODUCT_INDEX]
    corrected_nir = [correct_nir_pixel(pixel) for pixel in nir_product]
    gdal.write(corrected_nir, out_path)


if __name__ = "__main__":
    parser = argparse.ArgumentParser(description="Creates a geotif of corrected NIR pixels in product")
    parser.add_argument("in_path", help="Path to the HDF5 file to be processed")
    parser.add_argument("out_path", help="Location of the output geotif")
    args = parser.parse_args()
    save_nir_to_geotif(args.in_path, args.out_path)
```

- Argparse
  - Program can now be called from Bash
  - Can give description and help to users
- Loop replaced with **list comprehension**
  - More readable in this case
    - If you know what that is...

Ending

Every program needs the correct anatomy

Do not neglect the readme and environment

The human you are writing programs for is you, in the future

Show them the proper respect; they'll thank you for it

Futher exploration

Automated testing OR more reasons for functions

Taming matplotlib

Beyond gedit: Pycharm and Jupyter

Automatic documentation (yet more reason why functions are good)

Vectorising: Using Numpy properly

Further Reading

- Pocket Python
  - https://www.amazon.co.uk/dp/1449357016?tag=duc08-21&linkCode=osi&th=1&psc=1

- The Hitchiker's Guide to Python; section on code style
  - https://docs.python-guide.org/writing/style/

- Python docs themselves
  - https://docs.python.org/3/