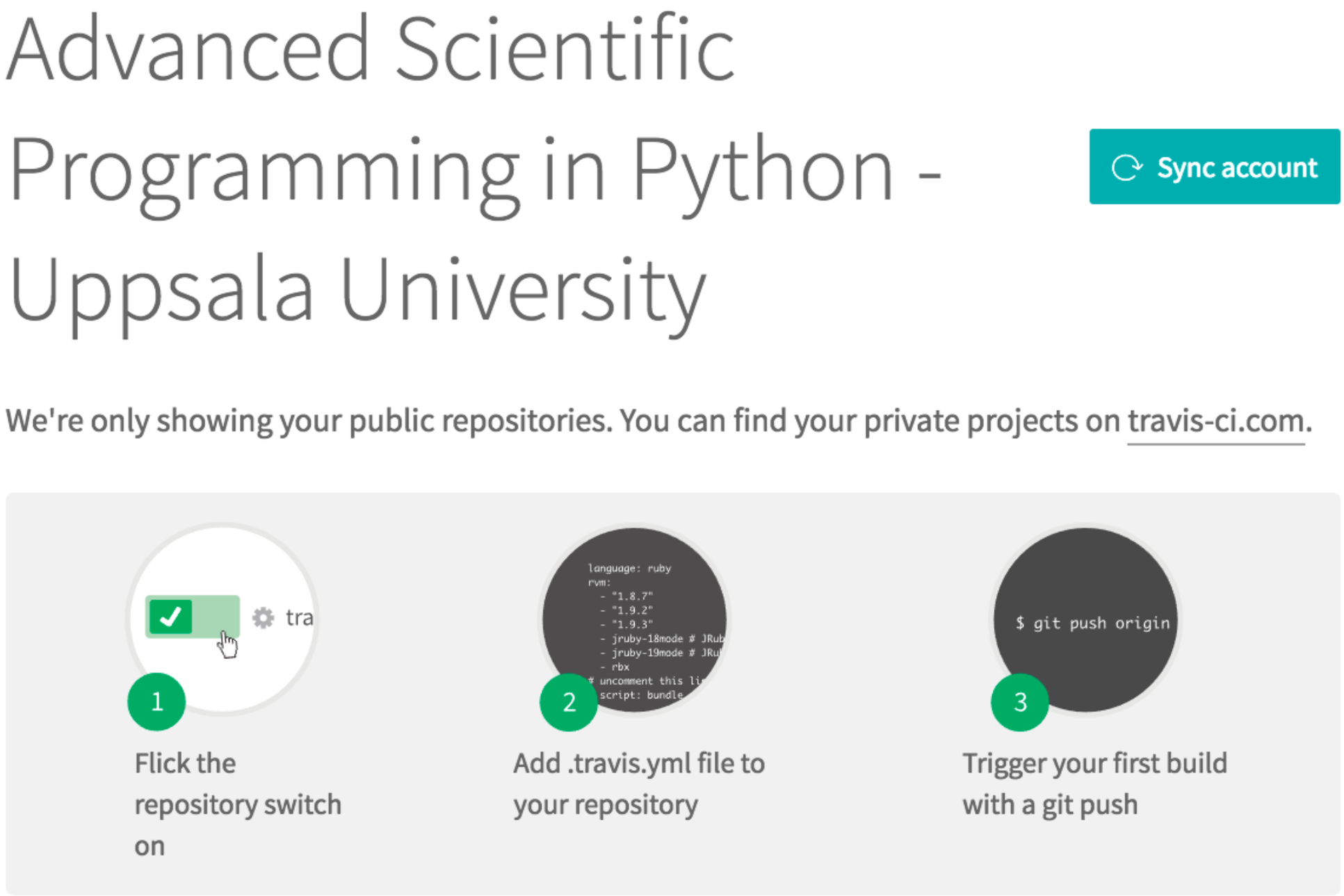
Travis CI

Enabling Travis CI for the repository



https://travis-ci.org/profile/your_username

Using Travis CI

- Travis CI has a strong integration with **`git`**
- I've placed our Fibonacci repository in GitHub
- Travis CI is controlled by a **`.travis.yml`** file in the repository. Here's a simple one:

```
# This project has python as the main language
language: python

# Run things with both these version of Python
python:
  - "2.7"
  - "3.4"

# Don't send email notifications for everything...
notifications:
  email: false

# We get a bare virtual machine
# We need to install our dependencies
install:
  - pip install numpy
  - pip install pytest

# Run the tests
script:
  - py.test
```

Travis CI build results

uu-python / travis-fib  

Current Branches Build History Pull Requests › Build #2 Job #2.1 More options

✓ **master** Use script not after_success for tests

Commit c67c5fd
Compare daee7ec..c67c5fd
Branch master

Filipe Maia authored and committed

#2.1 passed

Ran for 1 min 42 sec
about a minute ago

Restart job

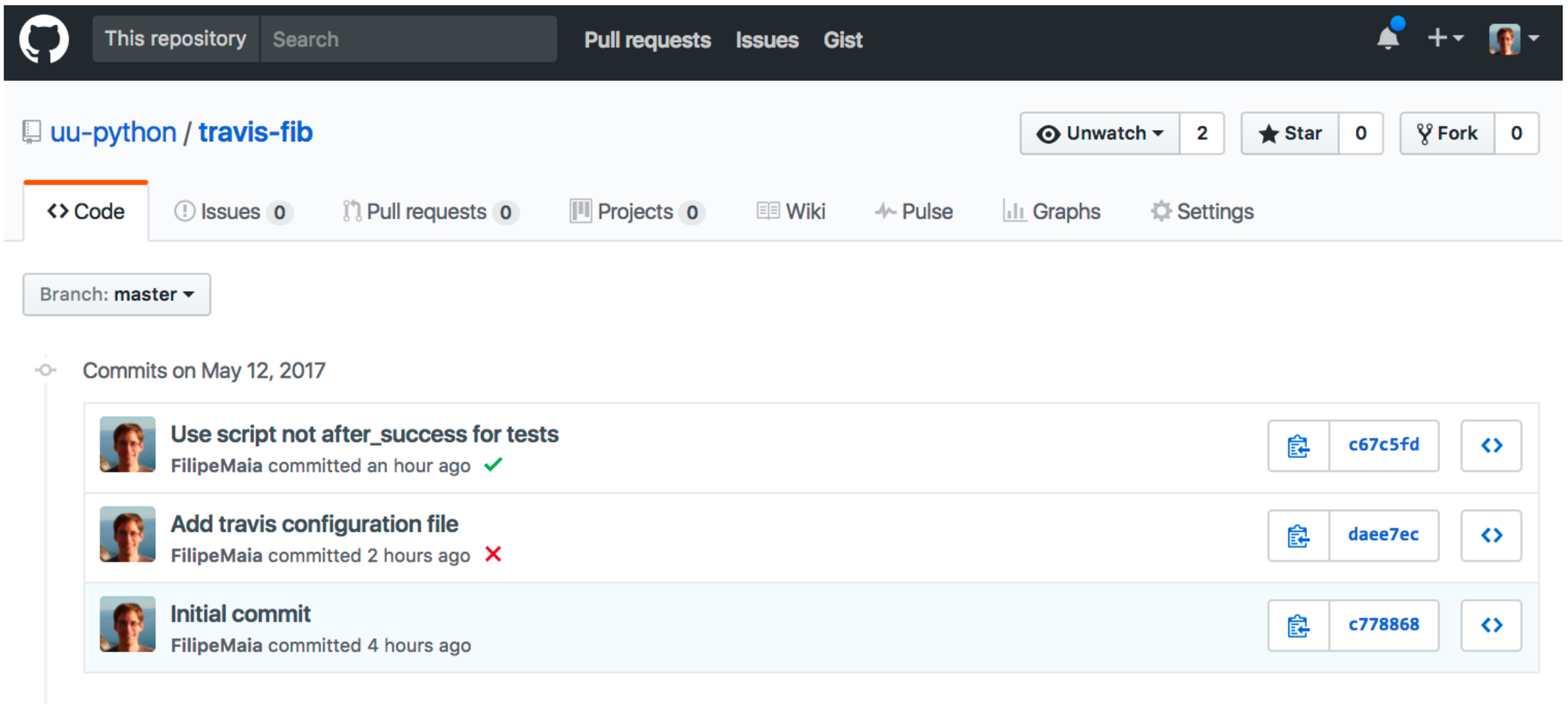
Job log View config

Remove log Raw log

```
▶   1 Worker information                                                    worker_info
▶   6 Build system information                                              system_info
   73
▶  74 $ export DEBIAN_FRONTEND=noninteractive                               fix.CVE-2015-7547
▶ 110 $ sudo apt-get install libssl1.0.0                                    update_libssl1.0.0
▶ 134 $ git clone --depth=50 --branch=master https://github.com/uu-python/travis-fib.git uu-python/travis-fib  git.checkout  1.89s
  144 $ source ~/virtualenv/python2.7/bin/activate                                                              0.00s
  145
  146 $ python --version
  147 Python 2.7.9
  148 $ pip --version
  149 pip 6.0.7 from /home/travis/virtualenv/python2.7.9/lib/python2.7/site-packages (python 2.7)
▶ 150 $ pip install numpy                                                   install.1  0.23s
▶ 153 $ pip install pytest                                                  install.2  0.21s
  157 $ py.test                                                                        0.27s
  158 ============================= test session starts ==============================
  159 platform linux2 -- Python 2.7.9 -- py-1.4.26 -- pytest-2.6.4
  160 collected 4 items
  161
  162 test_fib.py .
  163 test_fib_params.py ...
  164
  165 ========================== 4 passed in 0.02 seconds ===========================
  166
  167
  168 The command "py.test" exited with 0.
  169
  170 Done. Your build exited with 0.
```

Top

Travis CI Integration with GitHub



A More Complex Example

```
language: python

python:
  - "2.7"
  - "3.4"

notifications:
  email: false

sudo: false

cache:
  directories:
    - $HOME/.cache/pip
    - $HOME/arrayfire
    - $HOME/arrayfire-python
    - $HOME/local
```

A More Complex Example

```
addons:
  apt:
    sources:
      - ubuntu-toolchain-r-test
      - kubuntu-backports
    packages:
      - libatlas-base-dev
      - libfftw3-dev
      - gcc-4.9
      - g++-4.9
      - cmake
      - gdb
      - apport
```

A More Complex Example

```
before_install:
  - pip install codecov
  - cd $HOME
  - if [ ! -d "$HOME/arrayfire/.git" ]; then git clone  https://github.com/
arrayfire/arrayfire; else echo 'Using arrayfire from cached directory'; fi
  - mkdir -p arrayfire/build && cd arrayfire/build
  - git pull
  - cmake -DCMAKE_CXX_COMPILER=/usr/bin/g++-4.9 -DBUILD_CPU=ON -DBUILD_CUDA=OFF
-DBUILD_GRAPHICS=OFF -DBUILD_OPENCL=OFF -DBUILD_TEST=OFF -DBUILD_EXAMPLES=OFF
-DBUILD_UNIFIED=ON -DBUILD_CPU_ASYNC=ON -DCMAKE_INSTALL_PREFIX=${HOME}/local ..
  - make -j 2
  - make install
  - export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:${HOME}/local/lib
# Install arrayfire-python
  - cd $HOME
  - if [ ! -d "$HOME/arrayfire-python/.git" ]; then git clone  https://
github.com/arrayfire/arrayfire-python.git; else echo 'Using arrayfire-python
from cached directory'; fi
  - cd arrayfire-python
  - git pull
  - python setup.py install
```

A More Complex Example

```
install:
  - pip install numpy
  - pip install ipython
  - pip install pytest-cov
  - pip install pytest-benchmark
  - cd ${HOME}/build/FilipeMaia/afnumpy
  - python setup.py install

after_success:
  - codecov

before_script:
 # Set the core file limit to unlimited so a core file is generated upon
crash
 - ulimit -c unlimited -S
 - ulimit -c

script:
  - cd ${HOME}/build/FilipeMaia/afnumpy
  - coverage run --source afnumpy -m py.test --benchmark-skip  -v --color=yes
--showlocals --durations=10
  - python -m pytest -v --benchmark-only --benchmark-compare --benchmark-
autosave  --benchmark-group-by=fullname
  - for i in $(find ./ -maxdepth 1 -name 'core*' -print); do gdb python core*
-ex "thread apply all bt" -ex "set pagination 0" -batch; done;
```

Test Coverage

gh / FilipeMaia / afnumpy

Log in

Codecov
Code coverage done right.

Add tests showing shortcomings of dot(). See #24.

FilipeMaia 8 months ago ✔ CI Passed

5fa55ab master 6b77b6f

79.53% | ø | ø

Diff | Files | Build | Graphs

/ afnumpy

Files					Coverage
core	439	281	0	158	64.01%
lib	226	196	0	30	86.73%
linalg	95	47	0	48	49.47%
__init__.py	44	39	0	5	88.64%
decorators.py	60	58	0	2	96.67%
fft.py	77	74	0	3	96.10%
indexing.py	206	144	0	62	69.90%

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://github.com/numpy/numpy/blob/master/doc/HOWTO_DOCUMENT.rst.txt

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
```

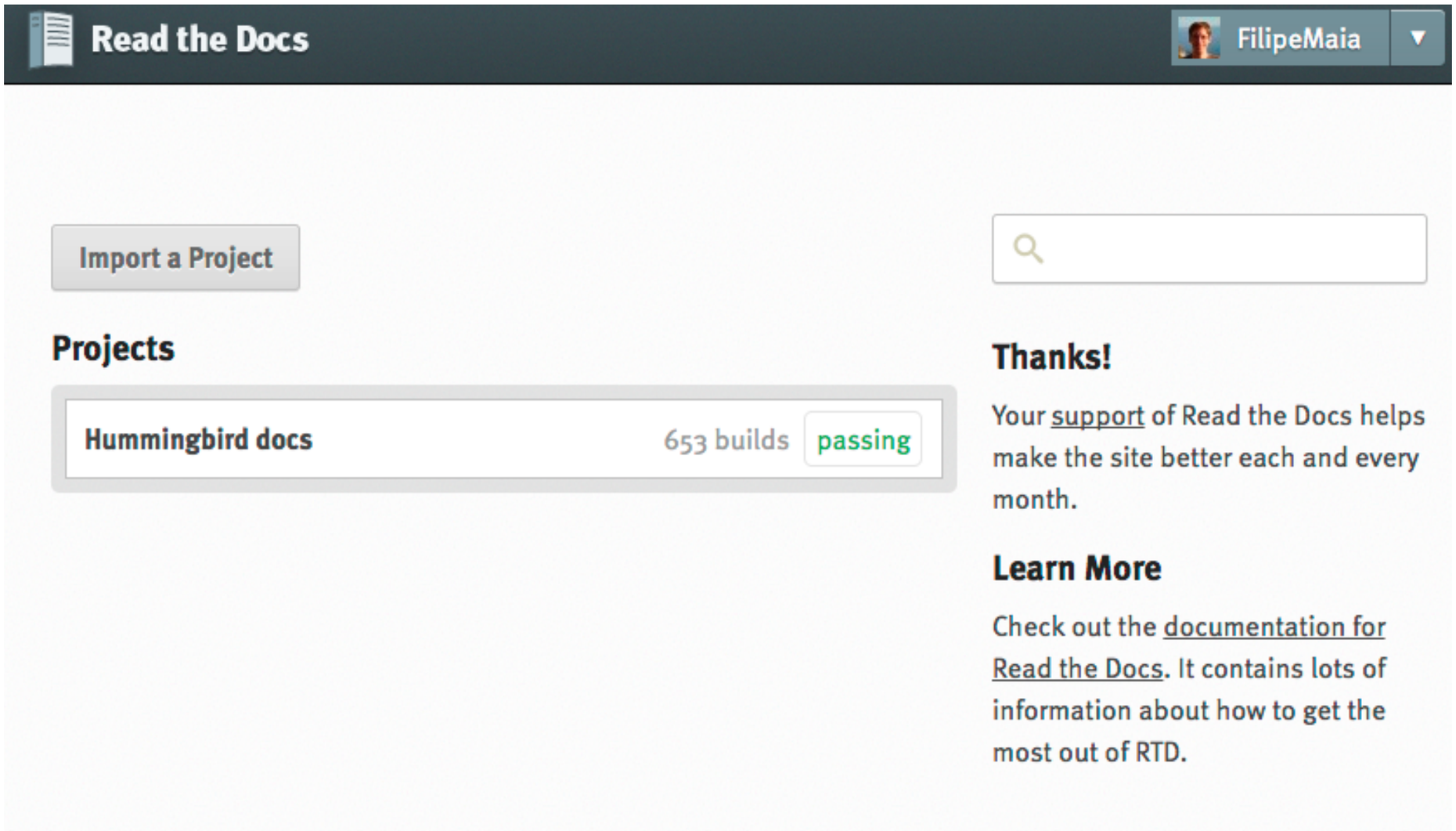
Indices and tables

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

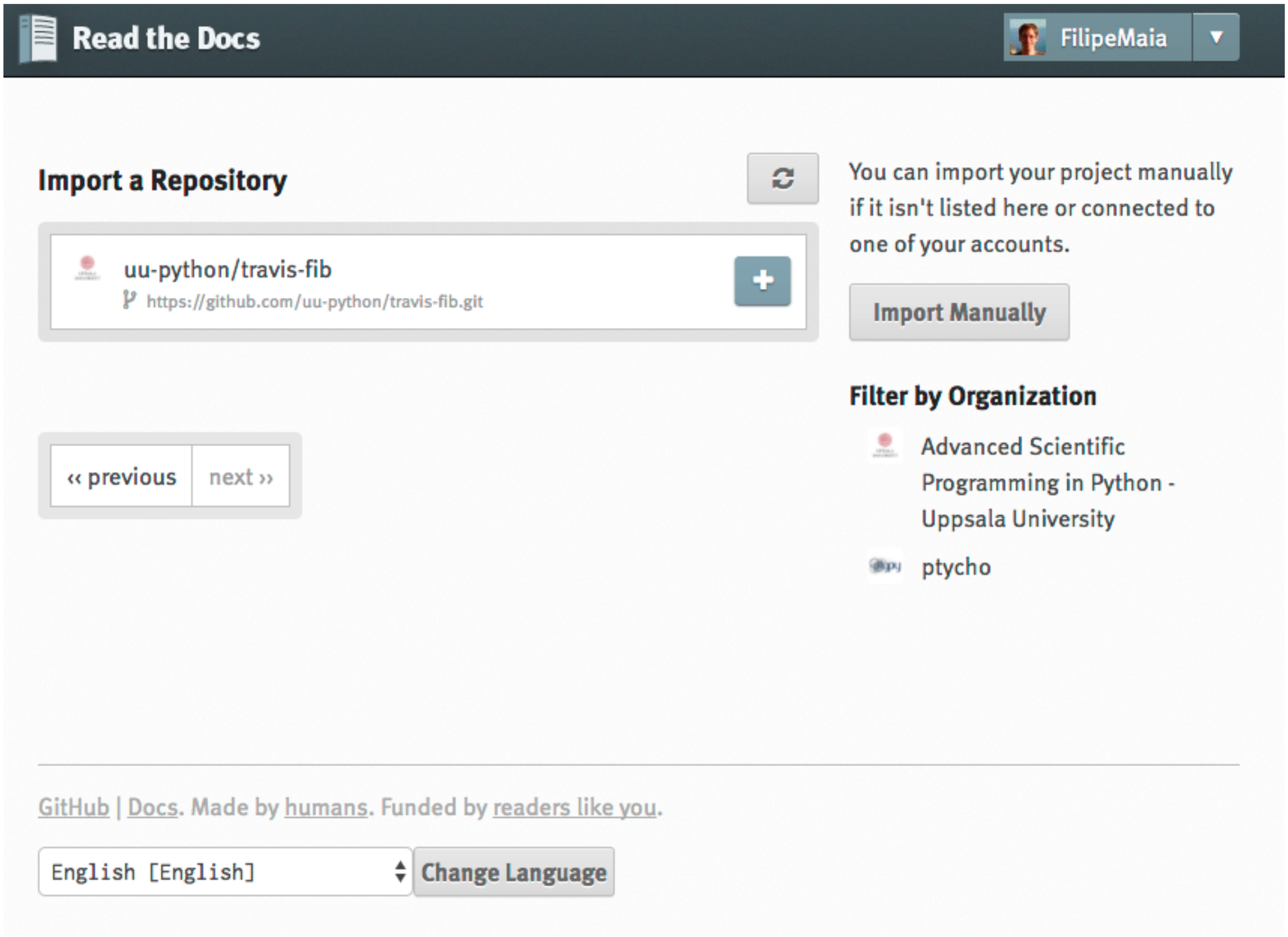
Adapted from:
http://docs.readthedocs.io/en/latest/getting_started.html

Read The Docs

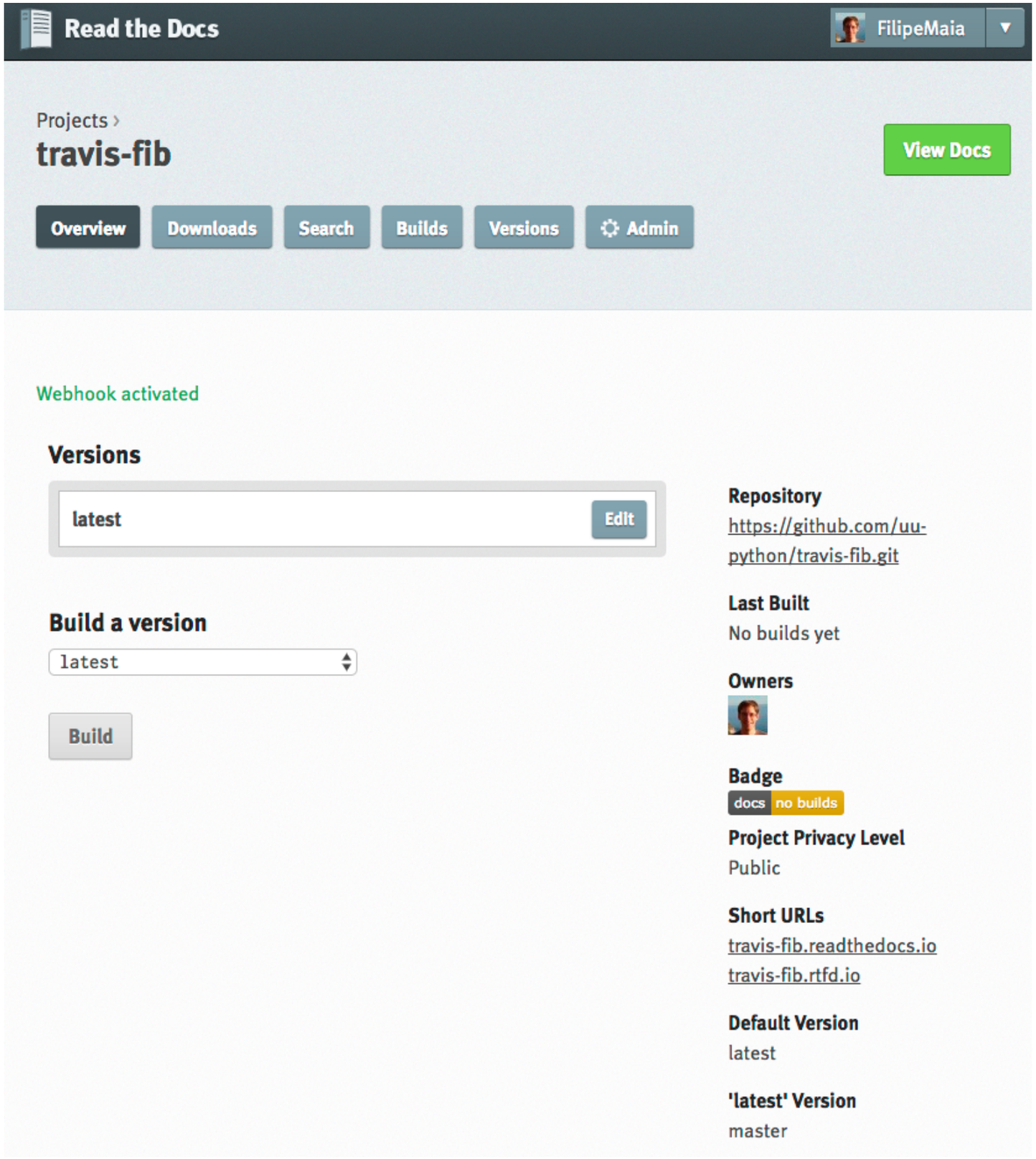
- Read The Docs (https://readthedocs.org) makes it easy to put your Sphinx documentation online.
- It's as simple as creating an account, connecting to your GitHub account and importing the project.



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.

Read The Docs

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

http://travis-fib.readthedocs.io/

Advantages and disadvantages of the multiple systems?

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 contains 108 031 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

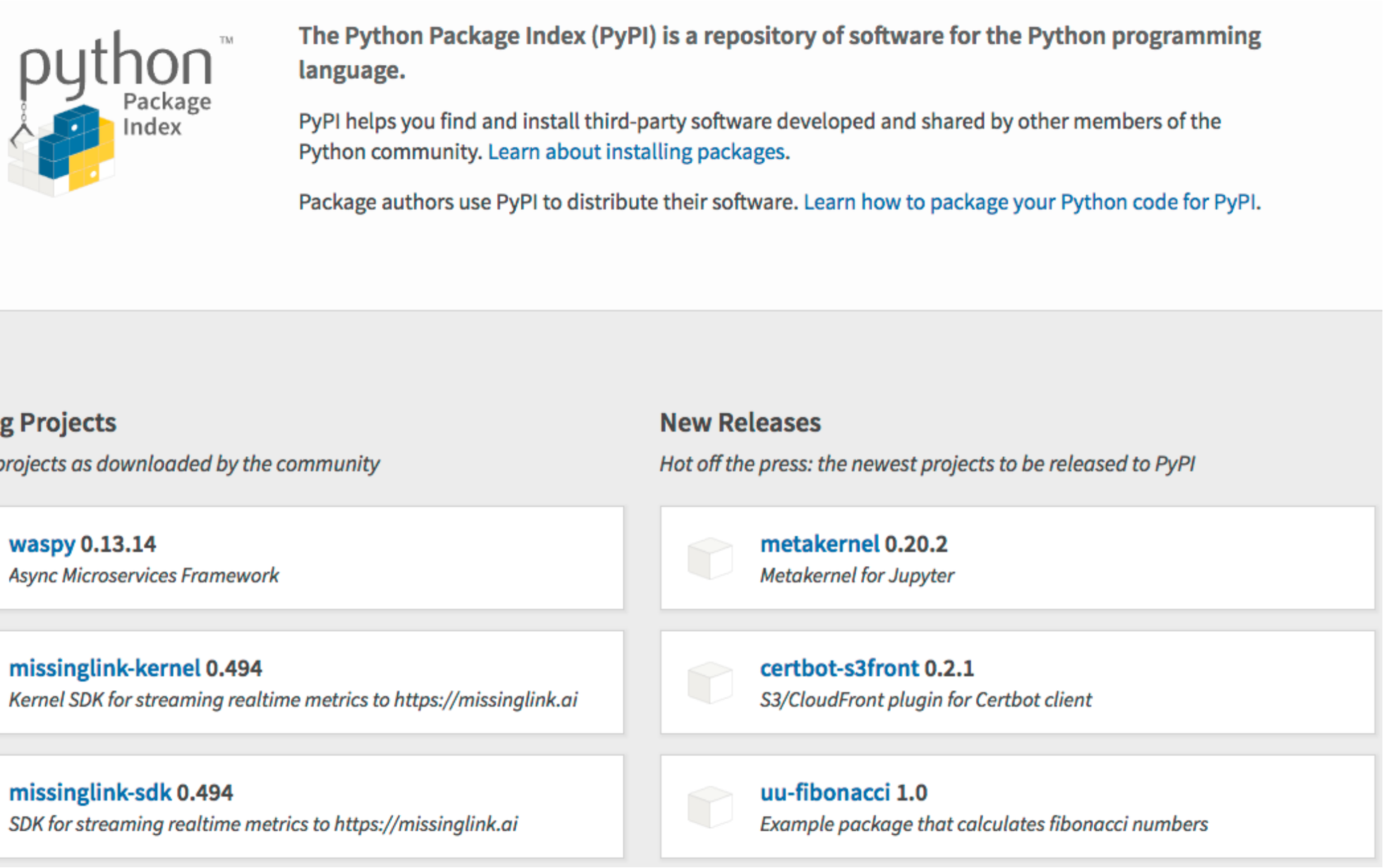
```
$ ipython2
In [1]: import fib

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/`




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!

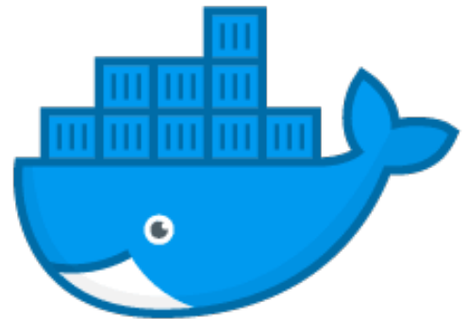
https://docs.conda.io/projects/conda-build/en/latest/index.html

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?

UPPSALA
UNIVERSITET

Data Containers: Efficient Memory Storage Using HDF5, And Pandas

Day 5

Advanced Scientific Programming with Python

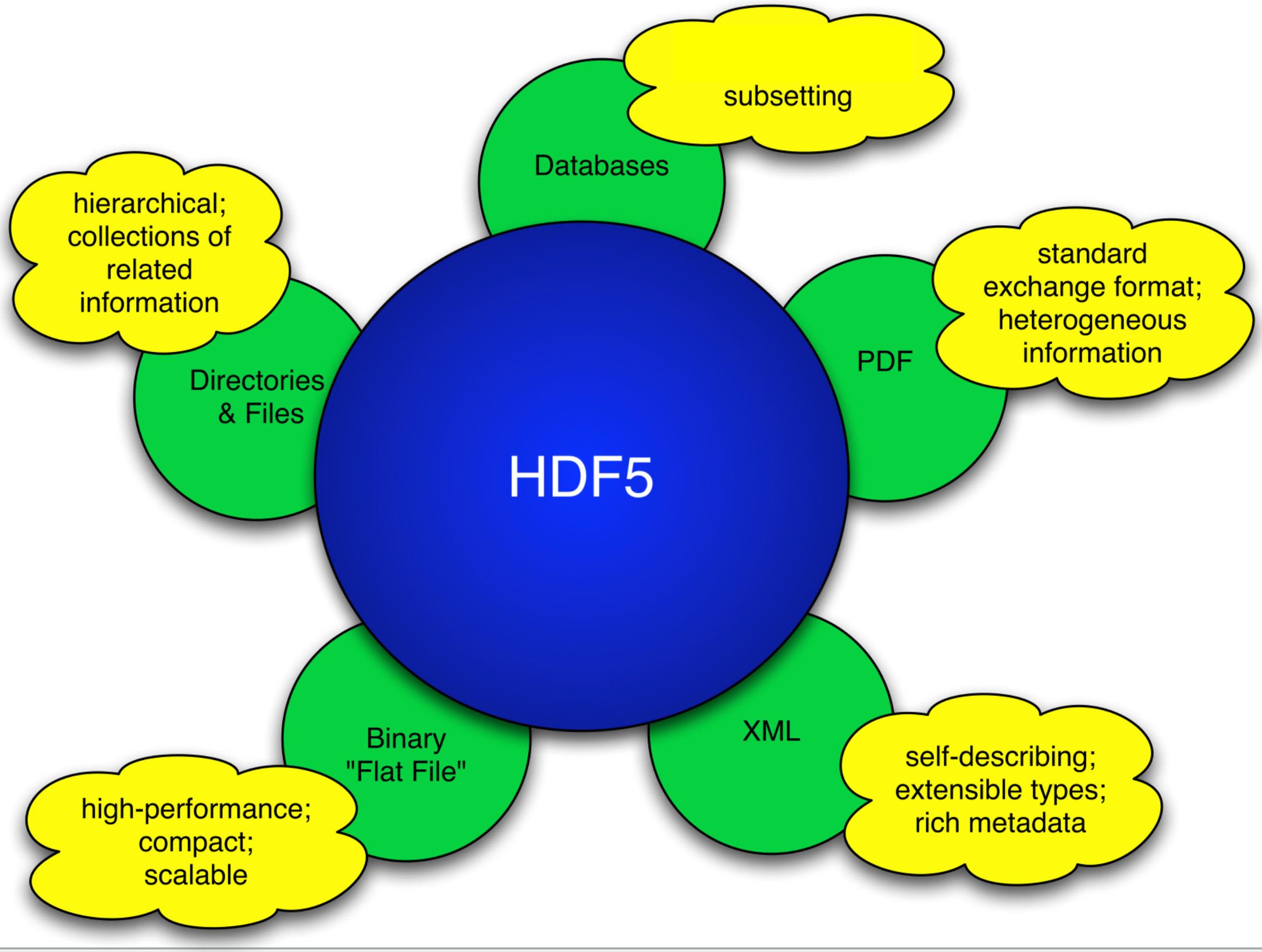
HDF5

Adapted from:
https://github.com/scopatz/hdf5-is-for-lovers/
https://support.hdfgroup.org/HDF5/doc/H5.intro.html
https://www.slideshare.net/HDFEOS/introduction-to-hdf5-data-and-programming-models-handson-exercise

What HDF5

- HDF5 stands for (H)eirarchical (D)ata (F)ormat (5)ive.
- It is supported by the HDFGroup.
- At its core HDF5 is binary file type specification.
- However, what makes HDF5 great is the numerous libraries written to interact with files of this type and their extremely rich feature set.
- Can represent complex data objects as well as associated metadata
- A **portable** file format with **no limit** on the number or size of data objects in the collection
- Free software (BSD, MIT kind of license)
- Implements a high-level API with C, C++, Fortran 90, and Java interfaces

HDF5 Has Characteristics Of …

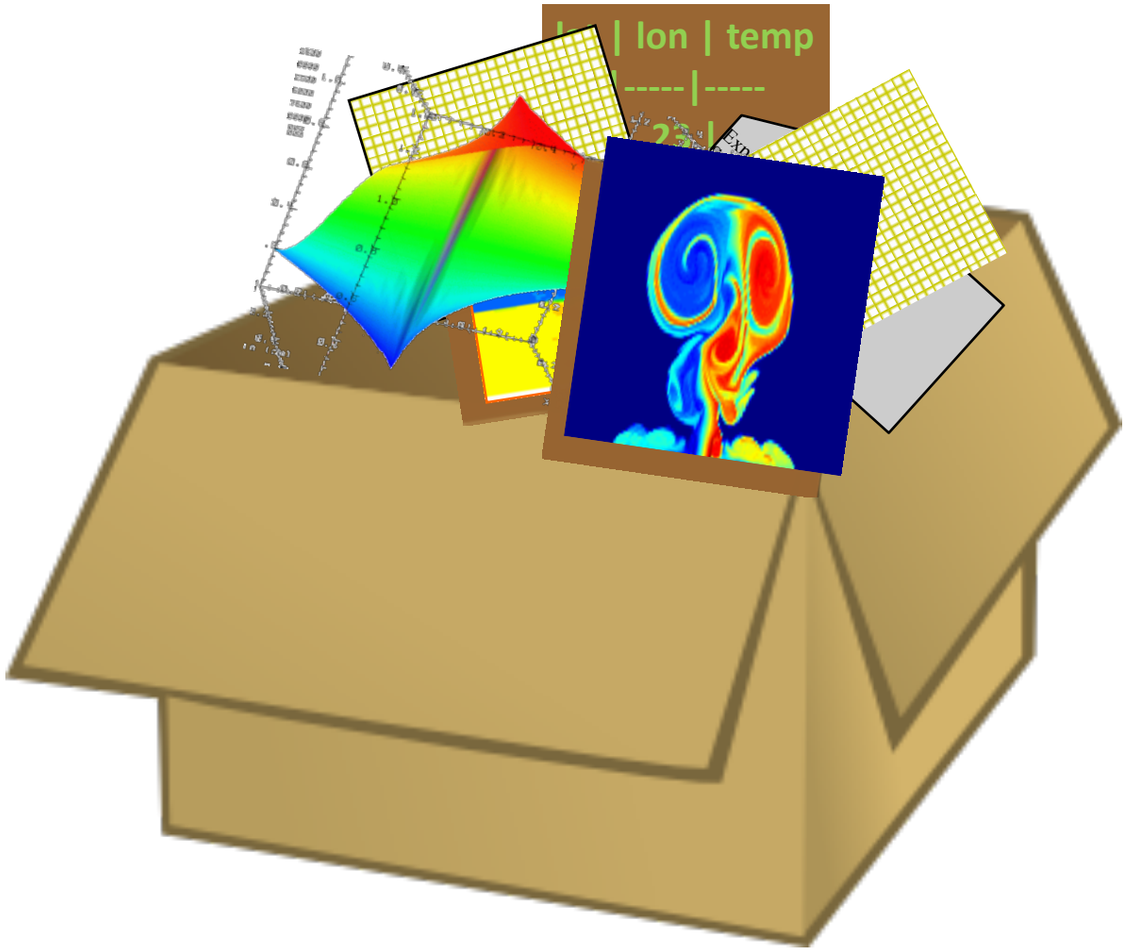


HDF5 Is Designed…

- for small or high volume and/or complex data
- for every size and type of system (portable)
- for flexible, efficient storage and I/O
- to enable applications to evolve in their use of HDF5 and to accommodate new models
- to support long-term data preservation
- Use it as a file format tool kit

HDF5 File

An HDF5 file is a **container** that holds data objects.



Intro to HDF5

- HDF5 files are organized in a hierarchical structure, with two primary structures: groups and datasets.

    1. **HDF5 group:** a grouping structure containing instances of zero or more groups or datasets, together with supporting metadata.

    2. **HDF5 dataset:** a multidimensional array of data elements, together with supporting metadata.

- Working with groups and group members is similar in many ways to working with directories and files in UNIX. As with UNIX directories and files, objects in an HDF5 file are often described by giving their full (or absolute) path names.

`/` signifies the root group.

`/foo` signifies a member of the root group called foo.

`/foo/zoo` signifies a member of the group foo, which in turn is a member of the root group.

- Any HDF5 group or dataset may have an associated attribute list. An HDF5 attribute is a user-defined HDF5 structure that provides extra information about an HDF5 object. Attributes are described in more detail below.

HDF5 Dataset

Metadata

Dataspace

Rank	Dimensions
3	Dim_1 = 4 Dim_2 = 5 Dim_3 = 7

Datatype

Integer

Properties

Chunked

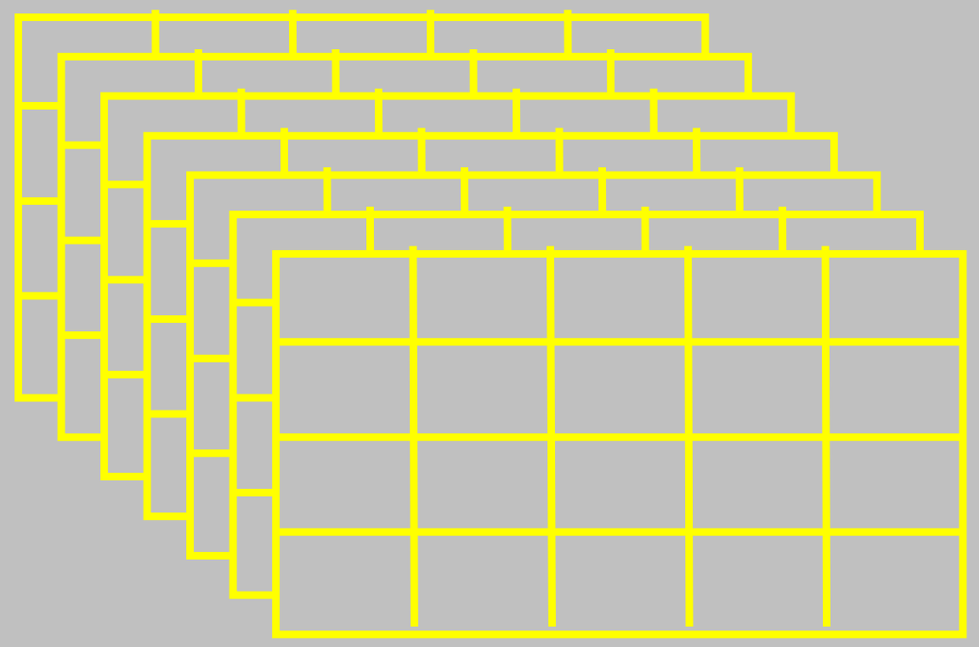
Compressed

(optional) Attributes

Time = 32.4

Pressure = 987

Data



Multi-dimensional array of identically typed data elements

- HDF5 datasets **organize and contain** "raw data values".
  - HDF5 datatypes describe individual data elements.
  - HDF5 dataspaces describe the logical layout of the data elements.

HDF5 Datasets

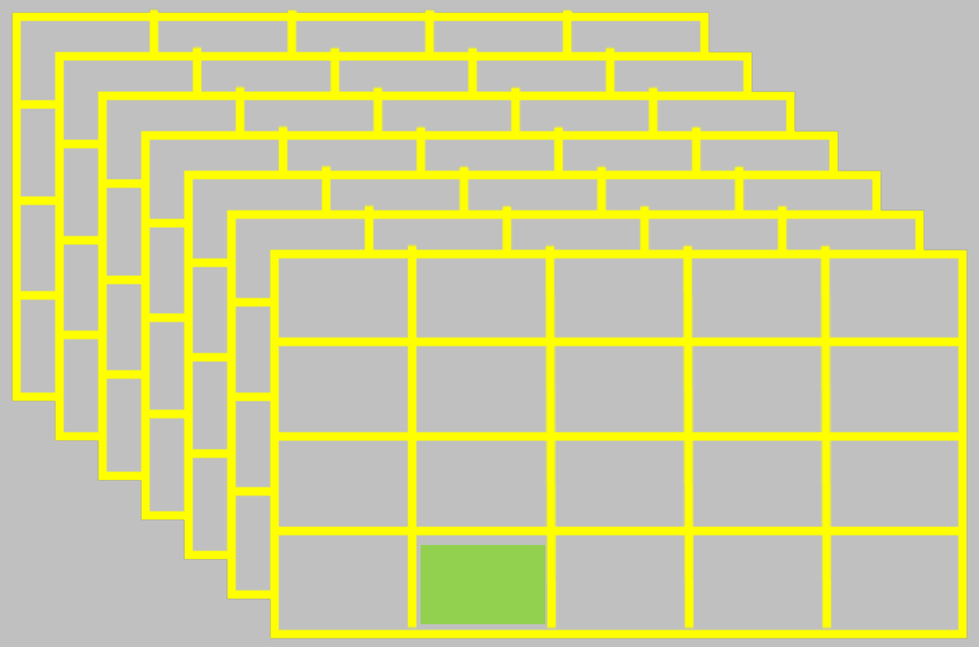
- A dataset is stored in a file in two parts: a header and a data array.
- There are four essential classes of information in any header: name, datatype, dataspace, and storage layout:
- Name. A dataset name is a sequence of alphanumeric ASCII characters.
- Datatype. Defines the kind of data stored in the dataset. Can be one of multiple builtin types (such as int, float, etc…) or user made compound datatypes (akin to a C struct).
- h5py (the main Python APIs for HDF5) supports the NumPy datatypes.

HDF5 Dataset & Datatype

HDF5 Datatype

Integer 32bit LE

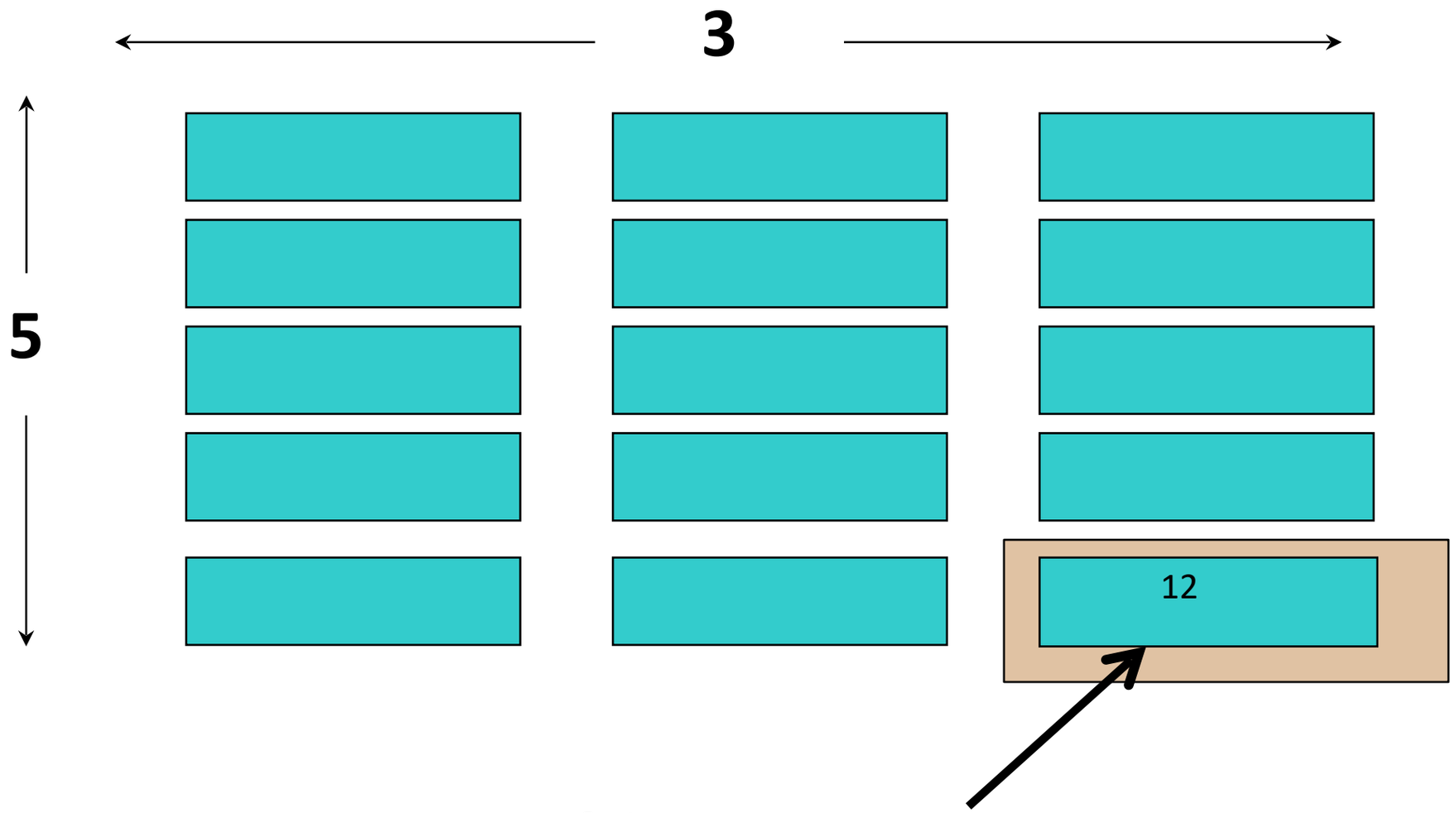
Specifications for single data element



Multi-dimensional array of identically typed data elements

- HDF5 datasets organize and contain “raw data values”.
  - HDF5 datatypes **describe individual data elements.**

HDF5 Dataset

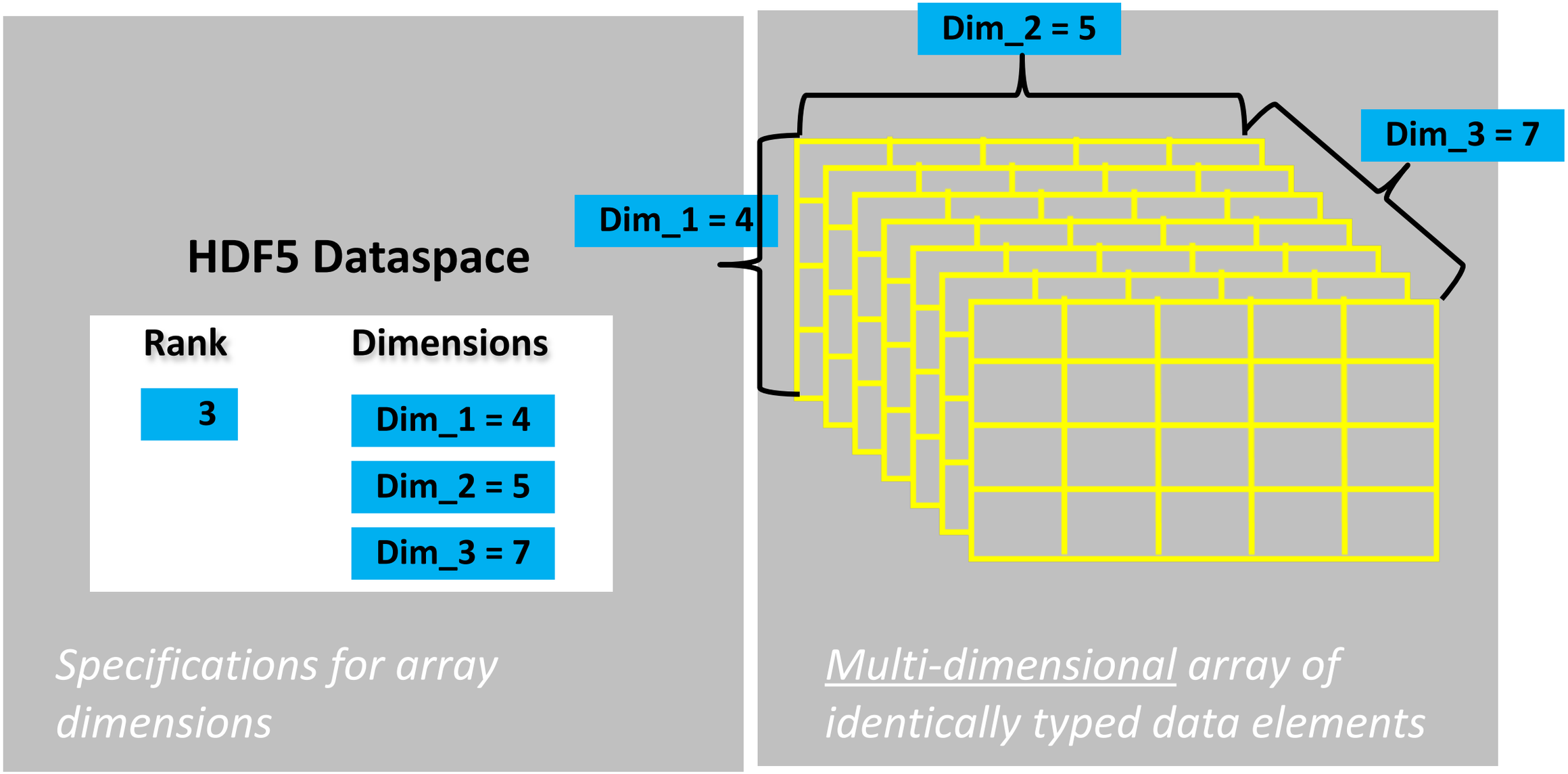


Datatype: **32-bit Integer**

Dataspace: **Rank = 2**

Dimensions = 5 x 3

HDF5 Dataset & Dataspace



- HDF5 datasets organize and contain “raw data values”.

  - HDF5 dataspaces **describe the logical layout of the data elements**

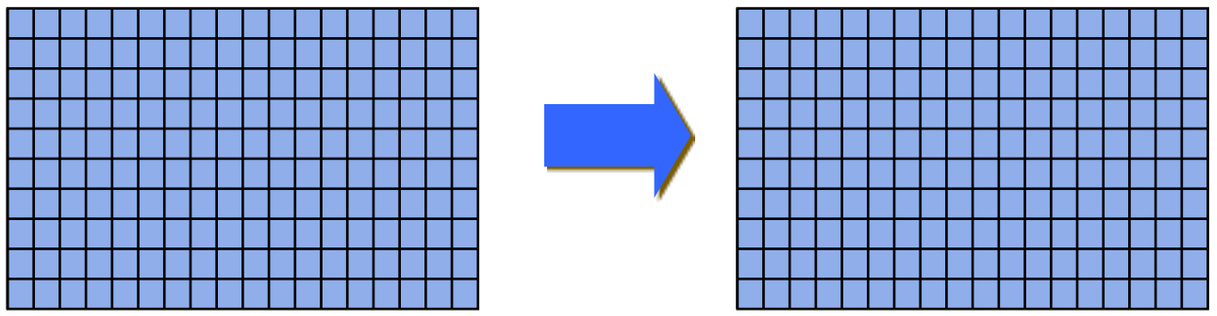
HDF5 Datasets

- Dataspace. A dataset dataspace describes the dimensionality of the dataset. The dimensions of a dataset can be fixed (unchanging), or they may be unlimited, which means that they are extendible (i.e. they can grow larger).
- Storage layout. The layout can be contiguous or chunked. In contiguous, the data is stored in the same linear way that it is organized in memory.
- Chunked storage involves dividing the dataset into equal-sized "chunks" that are stored separately. Chunking has three important benefits.

1. It makes it possible to achieve good performance when accessing subsets of the datasets, even when the subset to be chosen is orthogonal to the normal storage order of the dataset.
2. It makes it possible to compress large datasets and still achieve good performance when accessing subsets of the dataset.
3. It makes it possible efficiently to extend the dimensions of a dataset in any direction.

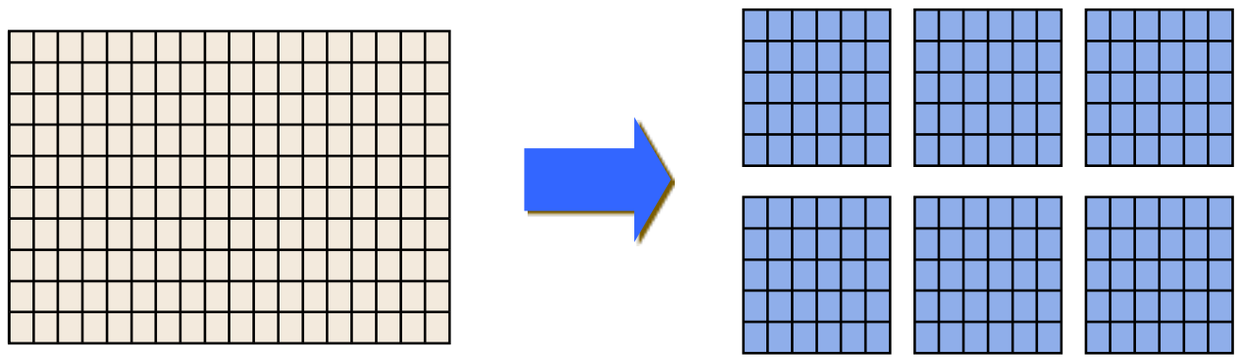
Dataset Storage Properties

Contiguous (default)



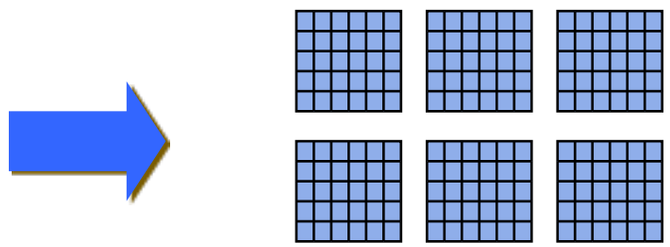
Data elements stored physically adjacent to each other

Chunked



Better access time for subsets; extendible

Chunked & Compressed



Improves storage efficiency, transmission speed

HDF5 Attributes

- Attributes are small named datasets that are attached to primary datasets, groups, or named datatypes.
- Attributes can be used to describe the nature and/or the intended usage of a dataset or group.
- An attribute has two parts: (1) a name and (2) a value. The value part contains one or more data entries of the same datatype.
- When accessing attributes, they can be identified by name or by an index value.
- The use of an index value makes it possible to iterate through all of the attributes associated with a given object.

HDF5 Attributes

- The HDF5 format and I/O library are designed with the assumption that attributes are small datasets.
- They are always stored in the object header of the object they are attached to.
- Because of this, large datasets should **not** be stored as attributes.
- How large is "large" is not defined by the library and is up to the user's interpretation. (Large datasets with metadata can be stored as supplemental datasets in a group with the primary dataset.)

h5py

- **h5py** and PyTables are the two most widely used HDF5 Python APIs.
- We'll start with **h5py**
- In **h5py** HDF5 Groups work like Python dictionaries, and datasets work like NumPy arrays
- Lets see how to create an HDF5 file:

```
>>> import h5py
>>> import numpy as np
>>>
>>> f = h5py.File("mytestfile.hdf5", "w")
```

- You'll end up with a File object which you can use to for example to create a new dataset:

```
>>> dset = f.create_dataset("mydataset", (100,), dtype='i')
```

h5py

- The object we created isn't an array, but an HDF5 dataset. Like NumPy arrays, datasets have both a shape and a data type:

```
>>> dset.shape
(100,)
>>> dset.dtype
dtype('int32')
```

- They also support array-style slicing. This is how you read and write data from a dataset in the file:

```
>>> dset[...] = np.arange(100)
>>> dset[0]
0
>>> dset[10]
10
>>> dset[0:100:10]
array([ 0, 10, 20, 30, 40, 50, 60, 70, 80, 90])
```

Groups and hierarchical organization

- Every object in an HDF5 file has a name, and they're arranged in a POSIX-style hierarchy with /-separators

```
>>> dset.name
u'/mydataset'
```

- Every object in an HDF5 file has a name, and they're arranged in a POSIX-style hierarchy with /-separators

- The "folders" in this system are called **groups**. The File object we created is itself a group, in this case the root group, named **/**:

```
>>> f.name
u'/'
```

- Creating a subgroup is accomplished via **create_group**

```
>>> grp = f.create_group("subgroup")
```

Groups and hierarchical organization

- All Group objects also have the **`create_*`** methods like **`File`**:

```
>>> dset2 = grp.create_dataset("another_dataset", (50,), dtype='f')
>>> dset2.name
u'/subgroup/another_dataset'
```

- By the way, you don't have to create all the intermediate groups manually. Specifying a full path works just fine:

```
>>> dset3 = f.create_dataset('subgroup2/dataset_three', (10,), dtype='i')
>>> dset3.name
u'/subgroup2/dataset_three'
```

- Groups support most of the Python dictionary-style interface. You retrieve objects in the file using the item-retrieval syntax:

```
>>> dataset_three = f[‘subgroup2/dataset_three']
```

- Iterating over a group provides the names of its members:

```
>>> for name in f:
...     print name
mydataset
subgroup
subgroup2
```

Groups and hierarchical organization

- Containership testing also uses names:

```
>>> "mydataset" in f
True
>>> "somethingelse" in f
False
```

- You can even use full path names:

```
>>> "subgroup/another_dataset" in f
True
```

- There are also the familiar **`keys()`**, **`values()`**, **`items()`** and **`iter()`** methods, as well as **`get()`**.

- Iterating over an entire file is accomplished with the Group methods **`visit()`** and **`visititems()`**, which takes a function:

```
>>> def printname(name):
...     print name
>>> f.visit(printname)
mydataset
subgroup
subgroup/another_dataset
...
```

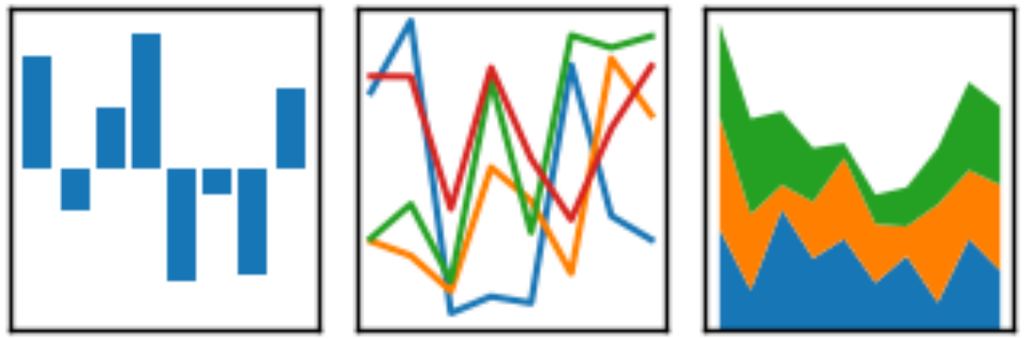
Attributes

- One of the best features of HDF5 is that you can store metadata right next to the data it describes.
- All groups and datasets support attached named bits of data called attributes.
- Attributes are accessed through the **`attrs`** proxy object, which again implements the dictionary interface:

```
>>> dset.attrs['temperature'] = 99.5
>>> dset.attrs['temperature']
99.5
>>> 'temperature' in dset.attrs
True
```

pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$



Adapted from:

Python for Data Analysis

http://shop.oreilly.com/product/0636920023784.do

Goals Of Pandas

- Data structures with labeled axes supporting automatic or explicit data alignment.
- This prevents common errors resulting from misaligned data and working with differently-indexed data coming from different sources.
- Integrated time series functionality.
- The same data structures handle both time series data and non-time series data.
- Arithmetic operations and reductions (like summing across an axis) would pass on the metadata (axis labels).
- Flexible handling of missing data.
- Merge and other relational operations found in popular database databases (SQLbased,for example).

Pandas Data Structures

- To get started with pandas, you will need to get comfortable with its two workhorse data structures: Series and DataFrame
- A Series is a one-dimensional array-like object containing an array of data (of any NumPy data type) and an associated array of data labels, called its index.
- The simplest Series is formed from only an array of data:

```
In [4]: obj = Series([4, 7, -5, 3])
In [5]: obj
Out[5]:
0 4
1 7
2 -5
3 3

In [6]: obj.values
Out[6]: array([ 4, 7, -5, 3])
In [7]: obj.index
Out[7]: Int64Index([0, 1, 2, 3])
```

Pandas Series

- Often it will be desirable to create a Series with an index identifying each data point:

```
In [8]: obj2 = Series([4, 7, -5, 3], index=['d', 'b', 'a', 'c'])
In [9]: obj2
Out[9]:
d 4
b 7
a -5
c 3
In [10]: obj2.index
Out[10]: Index([d, b, a, c], dtype=object)

In [11]: obj2['a']
Out[11]: -5
In [12]: obj2['d'] = 6
In [13]: obj2[['c', 'a', 'd']]
Out[13]:
c 3
a -5
d 6
```

Pandas Data Frames

- A DataFrame represents a tabular, spreadsheet-like data structure containing an ordered collection of columns, each of which can be a different value type (numeric, string, boolean, etc.).
- The DataFrame has both a row and column index; it can be thought of as a dict of Series (one for all sharing the same index).
- There are numerous ways to construct a DataFrame, though one of the most common is from a dict of equal-length lists or NumPy arrays

```
data = {'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada'],
 'year': [2000, 2001, 2002, 2001, 2002],
 'pop': [1.5, 1.7, 3.6, 2.4, 2.9]}
frame = DataFrame(data)

In [38]: frame
Out[38]:
 pop state year
0 1.5 Ohio 2000
1 1.7 Ohio 2001
2 3.6 Ohio 2002
3 2.4 Nevada 2001
4 2.9 Nevada 2002
```

Pandas Essential Functionality

- A critical method on pandas objects is **reindex**, which means to create a new object with the data conformed to a new index.

```
In [79]: obj = Series([4.5, 7.2, -5.3, 3.6], index=['d', 'b', 'a', 'c'])
In [80]: obj
Out[80]:
d 4.5
b 7.2
a -5.3
c 3.6

In [81]: obj2 = obj.reindex(['a', 'b', 'c', 'd', 'e'])
In [82]: obj2
Out[82]:
a -5.3
b 7.2
c 3.6
d 4.5
e NaN

In [83]: obj.reindex(['a', 'b', 'c', 'd', 'e'], fill_value=0)
Out[83]:
a -5.3
b 7.2
c 3.6
d 4.5
e 0.0
```

Pandas Essential Functionality

- Dropping entries from an axis (**drop**)

```
In [94]: obj = Series(np.arange(5.), index=['a', 'b', 'c', 'd', 'e'])
In [95]: new_obj = obj.drop('c')
In [96]: new_obj
Out[96]:
a 0
b 1
d 3
e 4
In [97]: obj.drop(['d', 'c'])
Out[97]:
a 0
b 1
e 43

In [98]: data = DataFrame(np.arange(16).reshape((4, 4)),
 ....: index=['Ohio', 'Colorado', 'Utah', 'New York'],
 ....: columns=['one', 'two', 'three', 'four'])
In [99]: data.drop(['Colorado', 'Ohio'])
Out[99]:
          one  two  three  four
Utah        8    9     10    11
New York   12   13     14    15
```

Pandas Essential Functionality

- Indexing, selection, and filtering
- Series indexing, selection and filtering (obj[...]) works analogously to NumPy arrays, except you can use the Series's index values instead of only integers.

```
In [102]: obj = Series(np.arange(4.), index=['a', 'b', 'c', 'd'])
In [103]: obj['b']
Out[103]: 1.0
In [104]: obj[1]
Out[104]: 1.0
```

- DataFrames also behave similarly

Pandas Essential Functionality

```
In [6]: data = DataFrame(np.arange(16).reshape((4, 4)),
   ...: index=['Ohio', 'Colorado', 'Utah', 'New York'],
   ...: columns=['one', 'two', 'three', 'four'])

In [7]: data
Out[7]:
          one  two  three  four
Ohio        0    1      2     3
Colorado    4    5      6     7
Utah        8    9     10    11
New York   12   13     14    15

In [8]: data['two']
Out[8]:
Ohio         1
Colorado     5
Utah         9
New York    13
Name: two, dtype: int64

In [9]: data[['three', 'one']]
Out[9]:
          three  one
Ohio          2    0
Colorado      6    4
Utah         10    8
New York     14   12
```

Pandas Essential Functionality

```
In [11]:  data[data['three'] > 5]
Out[11]:
          one  two  three  four
Colorado    4    5      6     7
Utah        8    9     10    11
New York   12   13     14    15

In [12]: data < 5
Out[12]:
            one    two  three   four
Ohio       True   True   True   True
Colorado   True  False  False  False
Utah      False  False  False  False
New York  False  False  False  False

In [13]: data[data < 5] = 0

In [14]: data
Out[14]:
          one  two  three  four
Ohio        0    0      0     0
Colorado    0    5      6     7
Utah        8    9     10    11
New York   12   13     14    15
```

Arithmetic And Data Alignment

- One of the most important pandas features is the behavior of arithmetic between objects with different indexes.
- When adding together objects, if any index pairs are not the same, the respective index in the result will be the union of the index pairs.

```
In [15]: s1 = Series([7.3, -2.5, 3.4, 1.5], index=['a', 'c', 'd', 'e'])

In [16]: s2 = Series([-2.1, 3.6, -1.5, 4, 3.1], index=['a', 'c', 'e', 'f',
'g'])
```

```
In [17]: s1
Out[17]:
a    7.3
c   -2.5
d    3.4
e    1.5
dtype: float64
```

```
In [18]: s2
Out[18]:
a   -2.1
c    3.6
e   -1.5
f    4.0
g    3.1
dtype: float64
```

```
In [19]: s1 + s2
Out[19]:
a    5.2
c    1.1
d    NaN
e    0.0
f    NaN
g    NaN
dtype: float64
```

Arithmetic And Data Alignment

- In arithmetic operations between differently-indexed objects, you might want to fill with a special value, like 0, when an axis label is found in one object but not the other:

```
In [20]: s1.add(s2, fill_value=0)
Out[20]:
a    5.2
c    1.1
d    3.4
e    0.0
f    4.0
g    3.1
dtype: float64
```

Statistics

- Operations in general exclude missing data

```
In [21]: data
Out[21]:
          one  two  three  four
Ohio        0    0      0     0
Colorado    0    5      6     7
Utah        8    9     10    11
New York   12   13     14    15

In [22]: data.mean()
Out[22]:
one      5.00
two      6.75
three    7.50
four     8.25
dtype: float64

In [23]: data.mean(1)
Out[23]:
Ohio         0.0
Colorado     4.5
Utah         9.5
New York    13.5
dtype: float64
```

Function Application

- You can apply arbitrary functions to your data

```
In [28]: data.apply(np.sqrt)
Out[28]:
                one       two     three      four
Ohio       0.000000  0.000000  0.000000  0.000000
Colorado   0.000000  2.236068  2.449490  2.645751
Utah       2.828427  3.000000  3.162278  3.316625
New York   3.464102  3.605551  3.741657  3.872983

In [27]: data.apply(np.cumsum)
Out[27]:
          one  two  three  four
Ohio        0    0      0     0
Colorado    0    5      6     7
Utah        8   14     16    18
New York   20   27     30    33

In [30]: data.apply(np.cumsum,1)
Out[30]:
          one  two  three  four
Ohio        0    0      0     0
Colorado    0    5     11    18
Utah        8   17     27    38
New York   12   25     39    54
```

Function Application

- You can apply arbitrary functions to your data

```
In [29]: data.apply(np.sum)
Out[29]:
one      20
two      27
three    30
four     33
dtype: int64
```

- You can apply NumPy element-wise functions directly

```
In [32]: np.sqrt(data)
Out[32]:
                one       two     three      four
Ohio       0.000000  0.000000  0.000000  0.000000
Colorado   0.000000  2.236068  2.449490  2.645751
Utah       2.828427  3.000000  3.162278  3.316625
New York   3.464102  3.605551  3.741657  3.872983
```

And Much Much More…

- Pandas is a very rich package, on par with NumPy
- There are excellent web resources to learn it
- For example http://pandas.pydata.org/
- I would also highly recommend "Python for Data Analysis" where most of the previous example were taken from.
- Written by the author of Pandas, but it covers much more!