

PYTHON FOR HPC

- Why use python?
- Some features of the python language
- Inner workings and performance
- Multithreading and multiprocessing examples
- A few words on concurrency

Repo: <https://github.com/appeltel/python-hpc-seminar>

(repo has links, code examples, etc...)

Why Python?

WHAT IS PYTHON?

- “Scripting” Language named after Monty Python
- Created in 1989 by Guido van Rossum, who is now the Benevolent Dictator for Life
- Multi-paradigm, with object-oriented and functional features
- CPython reference implementation (current version 3.6)
- Type “import this” for language philosophy

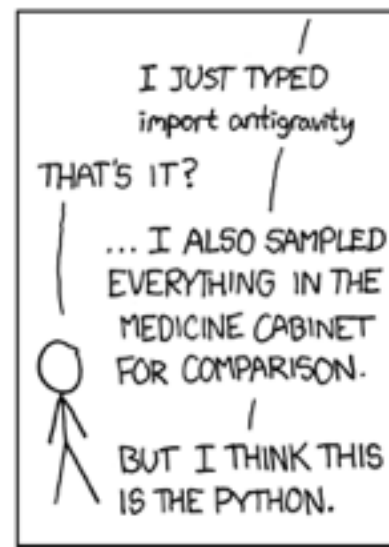
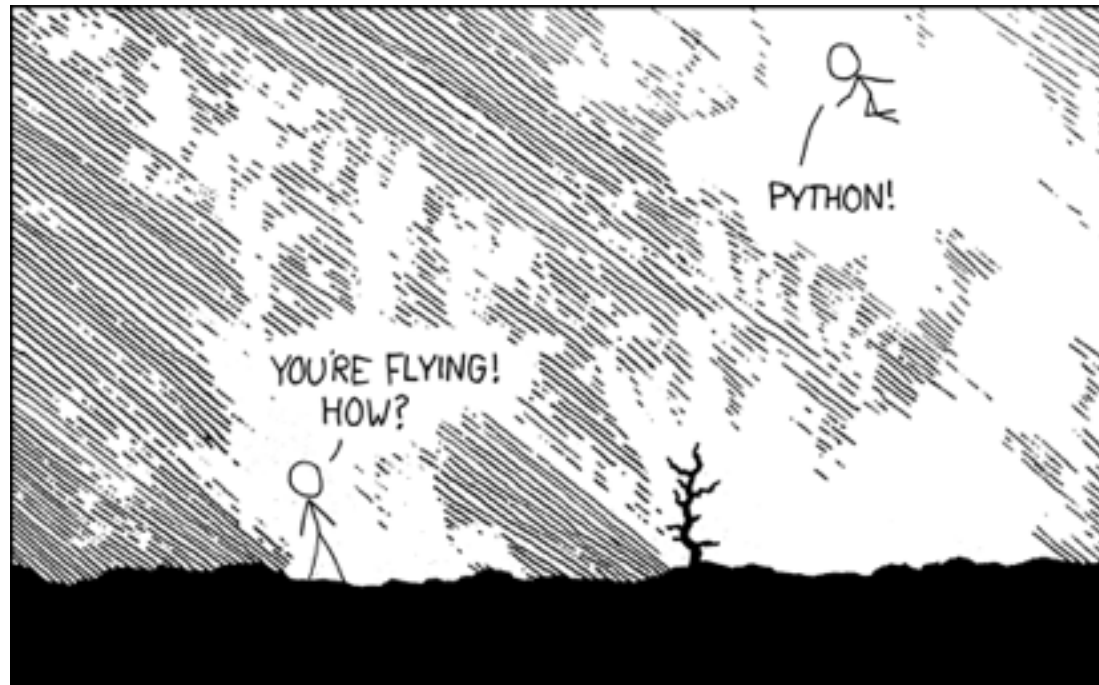


EASY TO LEARN, READ, AND USE

```
[>>> def fibonacci(n):  
[...     a, b = 0, 1  
[...     for _ in range(n):  
[...         a, b = b, a + b  
[...     return a  
[...  
[>>> fibonacci(7)  
13  
>>> █
```

- Built-in generic dynamic array (list) and mapping (dict) types
- Memory managed
- REPL, Notebooks, features for rapid prototyping
- BATTERIES INCLUDED

BATTERIES INCLUDED



- Full-featured standard library
- Extensive third-party packages
- PyPI service for distributing third party packages
- pip - tool for installing and managing packages
- virtualenv - tool for creating sandboxed environments for installing package sets



- Pre-packaged distribution of python and many scientific libraries and tools
- Tools for managing environments, installing yet more packages



django girls

PyOhio

About Python

- REPL - Read - Eval - Print - Loop
- Some basic things to explore on the REPL:
 - Names, assignment
 - Functions, modules, import system, namespaces
 - Dynamic typing, memory management
 - EAFP style - “We’re all responsible users here”
 - lists and dicts

MORE ON LISTS AND DICTS

- Built in list type “list” - (dynamic array)
- Built in mapping type “dict” - (hash table)
- Dynamic typing allows for easy representation of semi-structured data

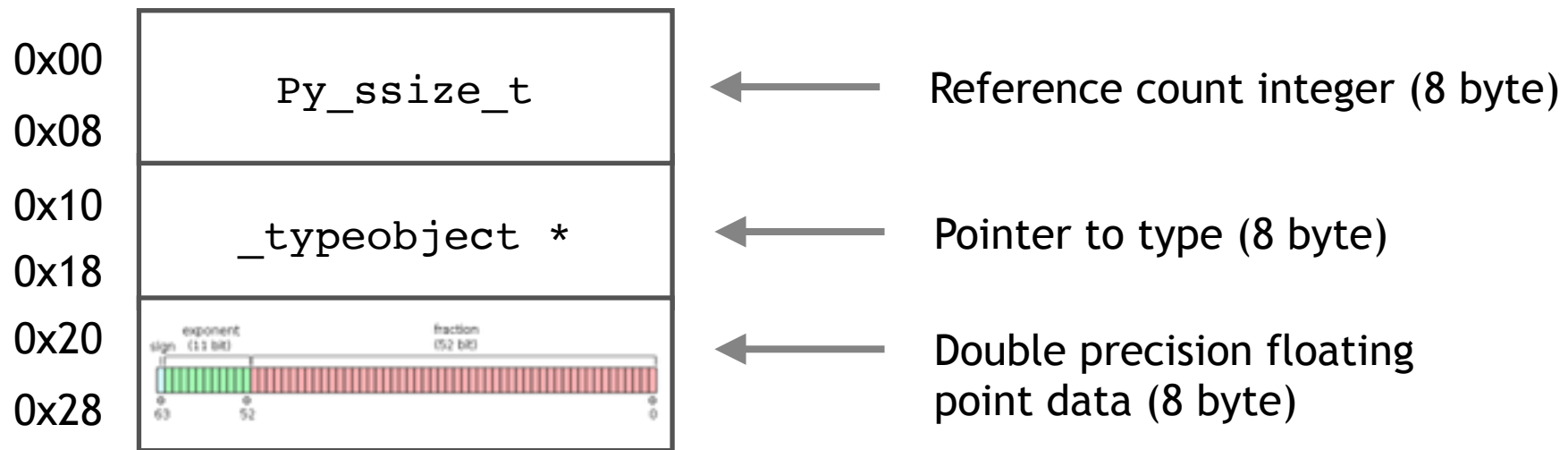
```
>>> record = {  
...     "name": "Eric Appelt",  
...     "email": "eric.appelt@gmail.com",  
...     "uid": 111694,  
...     "dogs": ["Abby", "Rocky"]  
... }  
>>> record['dogs'][1]  
'Rocky'  
>>>
```

How Python Works

- Each python object has a reference count
- Binding to a name, placement in a list position, increments the reference count
- When the reference count is zero, the object is destroyed and memory freed
- A garbage collector runs periodically to take care of circular references.

PYTHON OBJECTS IN MEMORY

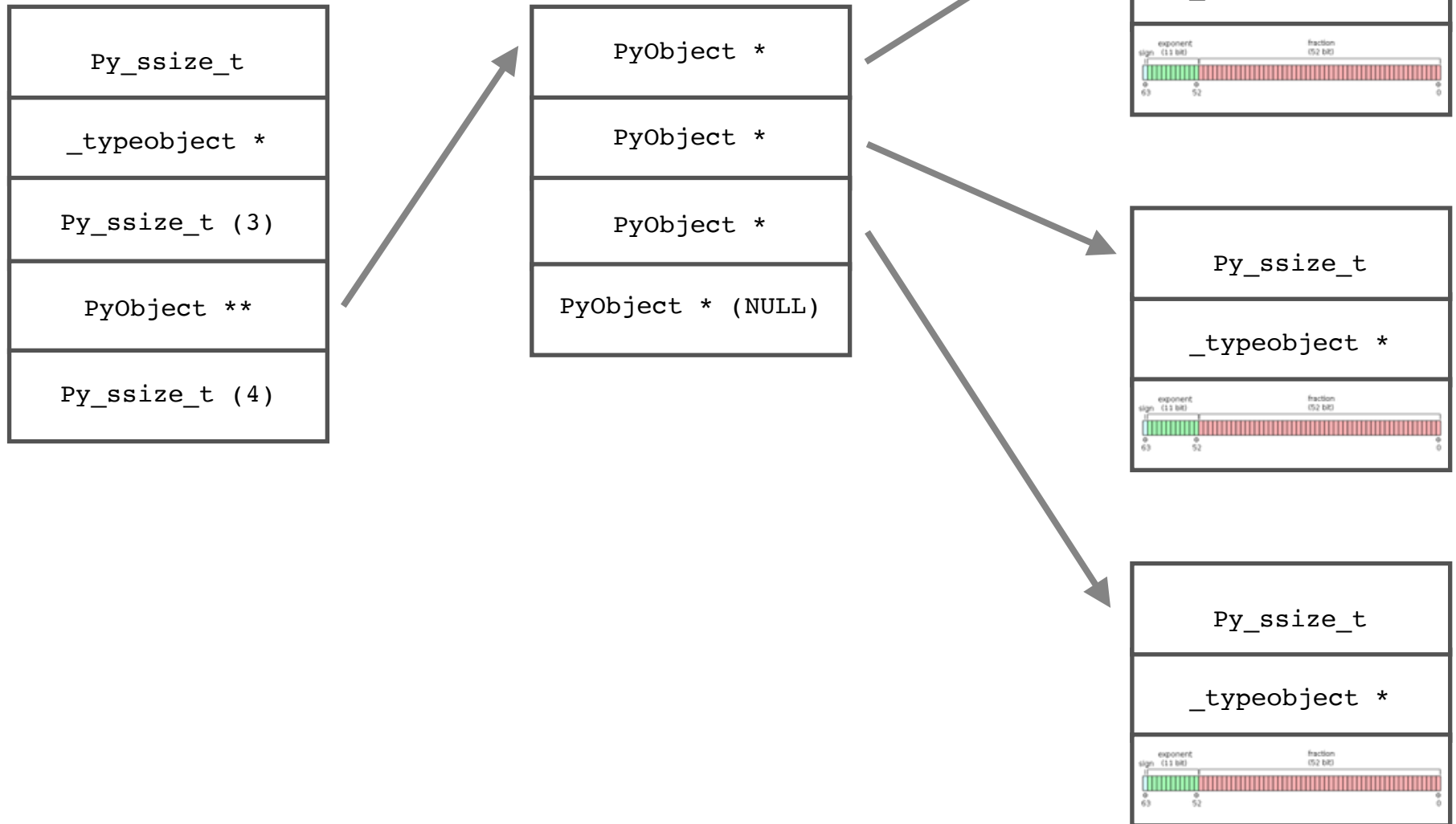
Python dynamic typing and memory management by reference counting incurs memory overhead (64-bit architecture):



A python “float” object.

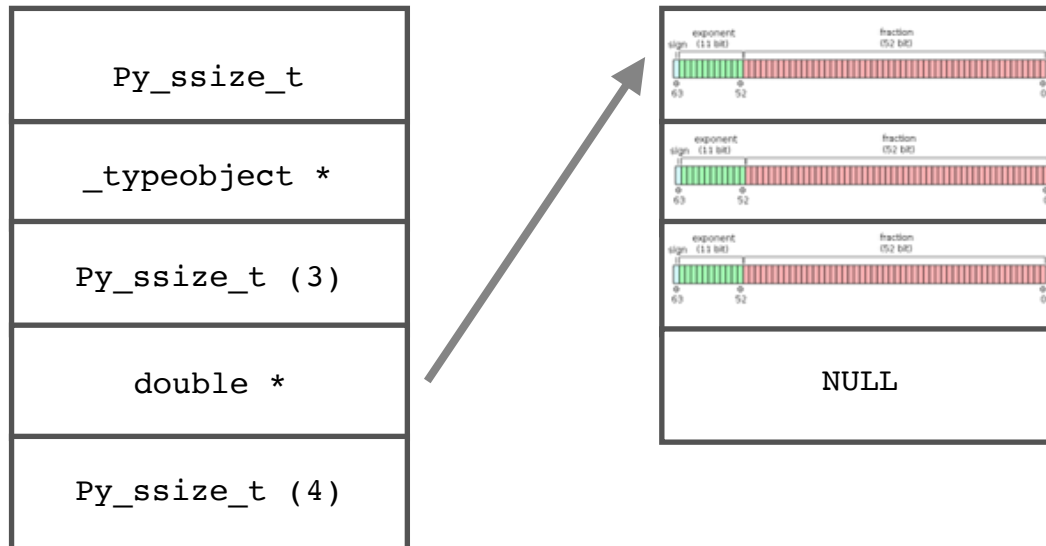
A PYTHON LIST OF FLOATS

A python list (dynamic array) of “floats”:



SOMETHING BETTER FOR HPC

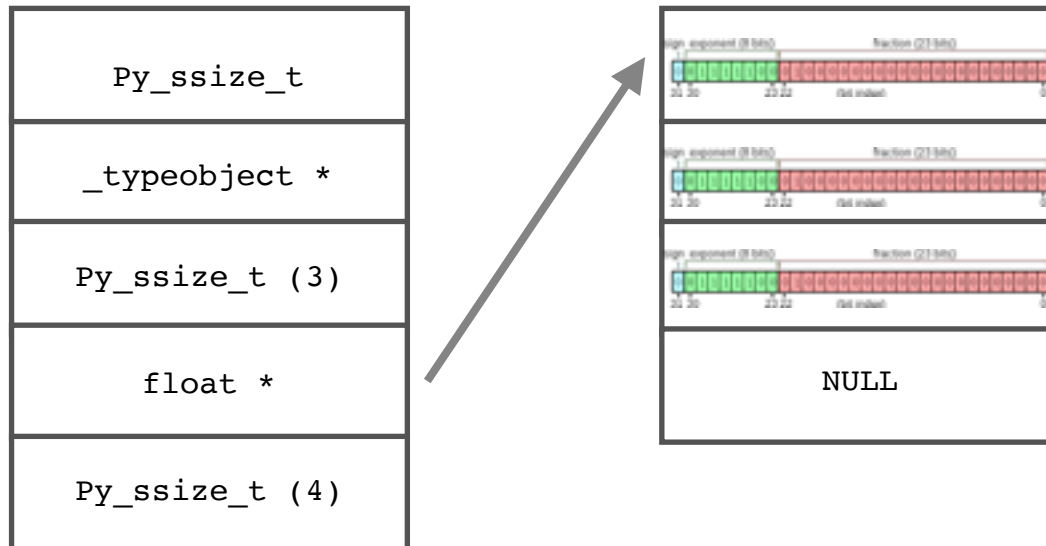
Something like this would be better for performance



8 bytes per double, no dereferencing individual pointers, checking types for each one.

SOMETHING BETTER FOR HPC

Or even better if you don't need double precision!



4 bytes per float, no
dereferencing individual
pointers, checking types
for each one.

- Python source is compiled into a platform-independent bytecode
- (CPython) A simple virtual stack machine, written in C, executes the bytecode
- (CPython) bytecode is NOT just-in-time compiled to native processor instructions
- Other implementations of python, such as pypy, do improve performance by compiling to native processor instructions.

EXAMPLE FUNCTION

```
>>> import dis
>>> def foo(x, y):
...     return x * y + 17
...
>>> foo.__code__.co_varnames
('x', 'y')
>>> foo.__code__.co_consts
(None, 17)
>>> [hex(b) for b in foo.__code__.co_code]
['0x7c', '0x0', '0x7c', '0x1', '0x14', '0x0', '0x64', '0x1', '0x17', '0x0', '0x53', '0x0']
>>> dis.dis(foo)
2          0 LOAD_FAST           0 (x)
          2 LOAD_FAST           1 (y)
          4 BINARY_MULTIPLY
          6 LOAD_CONST           1 (17)
          8 BINARY_ADD
         10 RETURN_VALUE

>>> █
```

EXAMPLE FUNCTION INVOCATION

CALLED: foo(3, 5)

function code

0	LOAD_FAST	0 (x)
2	LOAD_FAST	1 (y)
4	BINARY_MULTIPLY	
6	LOAD_CONST	1 (17)
8	BINARY_ADD	
10	RETURN_VALUE	

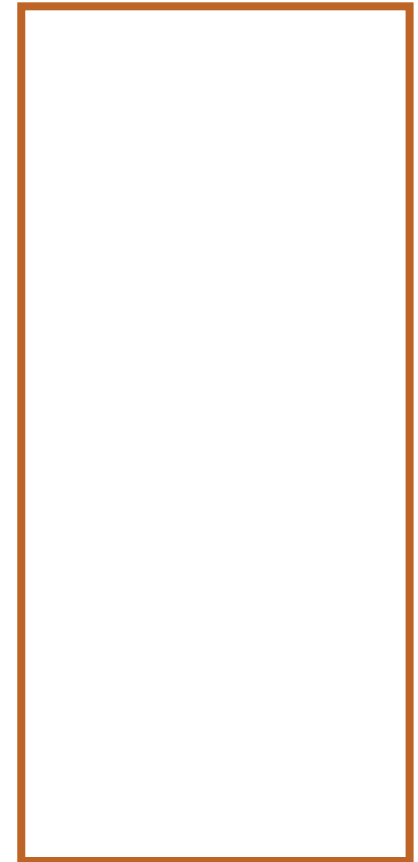
co_varnames

PyObject * (3)	PyObject * (5)
----------------	----------------

co_consts

PyObject * (None)	PyObject * (17)
-------------------	-----------------


object stack



EXAMPLE FUNCTION INVOCATION

CALLED: foo(3, 5)

function code



0	LOAD_FAST	0 (x)
2	LOAD_FAST	1 (y)
4	BINARY_MULTIPLY	
6	LOAD_CONST	1 (17)
8	BINARY_ADD	
10	RETURN_VALUE	

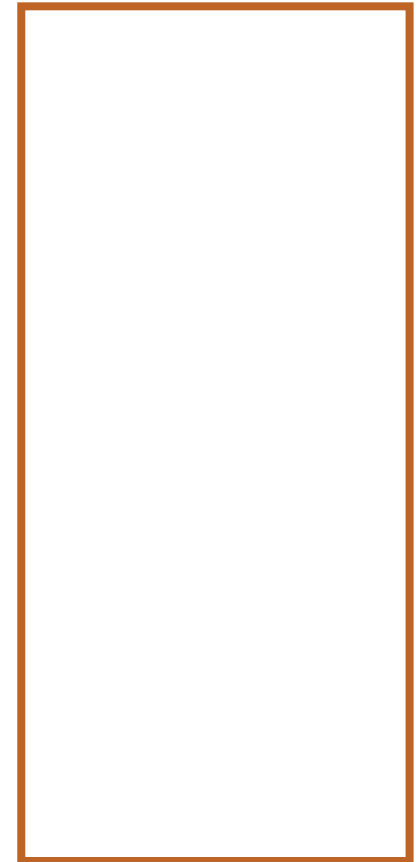
co_varnames

PyObject * (3)	PyObject * (5)
----------------	----------------

co_consts

PyObject * (None)	PyObject * (17)
-------------------	-----------------


object stack



EXAMPLE FUNCTION INVOCATION

CALLLED: foo(3, 5)

function code



0	LOAD_FAST	0 (x)
2	LOAD_FAST	1 (y)
4	BINARY_MULTIPLY	
6	LOAD_CONST	1 (17)
8	BINARY_ADD	
10	RETURN_VALUE	

co_varnames

PyObject * (3)	PyObject * (5)
----------------	----------------

co_consts

PyObject * (None)	PyObject * (17)
-------------------	-----------------

PUSH!

object stack




PyObject * (3)

EXAMPLE FUNCTION INVOCATION

CALLLED: foo(3, 5)

function code



0	LOAD_FAST	0 (x)
2	LOAD_FAST	1 (y)
4	BINARY_MULTIPLY	
6	LOAD_CONST	1 (17)
8	BINARY_ADD	
10	RETURN_VALUE	

co_varnames

PyObject * (3)	PyObject * (5)
----------------	----------------

co_consts

PyObject * (None)	PyObject * (17)
-------------------	-----------------


object stack

PyObject * (3)

EXAMPLE FUNCTION INVOCATION

CALLLED: foo(3, 5)

function code



0	LOAD_FAST	0 (x)
2	LOAD_FAST	1 (y)
4	BINARY_MULTIPLY	
6	LOAD_CONST	1 (17)
8	BINARY_ADD	
10	RETURN_VALUE	

co_varnames

PyObject * (3)	PyObject * (5)
----------------	----------------

co_consts

PyObject * (None)	PyObject * (17)
-------------------	-----------------

PUSH!

object stack




PyObject * (5)
PyObject * (3)

EXAMPLE FUNCTION INVOCATION

CALLLED: foo(3, 5)

function code



0	LOAD_FAST	0 (x)
2	LOAD_FAST	1 (y)
4	BINARY_MULTIPLY	
6	LOAD_CONST	1 (17)
8	BINARY_ADD	
10	RETURN_VALUE	

co_varnames

PyObject * (3)	PyObject * (5)
----------------	----------------

co_consts

PyObject * (None)	PyObject * (17)
-------------------	-----------------

object stack

PyObject * (5)


PyObject * (3)

EXAMPLE FUNCTION INVOCATION

CALLER: foo(3, 5)

function code

0	LOAD_FAST	0 (x)
2	LOAD_FAST	1 (y)
4	BINARY_MULTIPLY	
6	LOAD_CONST	1 (17)
8	BINARY_ADD	
10	RETURN_VALUE	



co_varnames

PyObject * (3)	PyObject * (5)
----------------	----------------

co_consts

PyObject * (None)	PyObject * (17)
-------------------	-----------------

object stack

PyObject * (5)

POP!




PyObject * (3)

EXAMPLE FUNCTION INVOCATION

CALLLED: foo(3, 5)

function code



0	LOAD_FAST	0 (x)
2	LOAD_FAST	1 (y)
4	BINARY_MULTIPLY	
6	LOAD_CONST	1 (17)
8	BINARY_ADD	
10	RETURN_VALUE	

co_varnames

PyObject * (3)	PyObject * (5)
----------------	----------------

co_consts

PyObject * (None)	PyObject * (17)
-------------------	-----------------

object stack

PyObject * (5)

PyObject * (3)

POP !




EXAMPLE FUNCTION INVOCATION

CALLLED: `foo(3, 5)`

function code

0	LOAD_FAST	0 (x)
2	LOAD_FAST	1 (y)
4	BINARY_MULTIPLY	
6	LOAD_CONST	1 (17)
8	BINARY_ADD	
10	RETURN_VALUE	



co_varnames

PyObject * (3)	PyObject * (5)
----------------	----------------

co_consts

PyObject * (None)	PyObject * (17)
-------------------	-----------------

`__mult__`

PyObject * (5)

PyObject * (3)



PyObject * (15)


object stack



EXAMPLE FUNCTION INVOCATION

CALLLED: foo(3, 5)

function code



0	LOAD_FAST	0 (x)
2	LOAD_FAST	1 (y)
4	BINARY_MULTIPLY	
6	LOAD_CONST	1 (17)
8	BINARY_ADD	
10	RETURN_VALUE	

co_varnames

PyObject * (3)	PyObject * (5)
----------------	----------------

co_consts

PyObject * (None)	PyObject * (17)
-------------------	-----------------

object stack

PyObject * (15)

PyObject * (15)




PUSH!

EXAMPLE FUNCTION INVOCATION

CALLLED: foo(3, 5)

function code



0	LOAD_FAST	0 (x)
2	LOAD_FAST	1 (y)
4	BINARY_MULTIPLY	
6	LOAD_CONST	1 (17)
8	BINARY_ADD	
10	RETURN_VALUE	

co_varnames

PyObject * (3)	PyObject * (5)
----------------	----------------

co_consts

PyObject * (None)	PyObject * (17)
-------------------	-----------------


object stack

PyObject * (15)

EXAMPLE FUNCTION INVOCATION

CALLLED: `foo(3, 5)`

function code



0	LOAD_FAST	0 (x)
2	LOAD_FAST	1 (y)
4	BINARY_MULTIPLY	
6	LOAD_CONST	1 (17)
8	BINARY_ADD	
10	RETURN_VALUE	


co_varnames

PyObject * (3)	PyObject * (5)
----------------	----------------

co_consts

PyObject * (None)	PyObject * (17)
-------------------	-----------------

object stack



PUSH!


PyObject * (17)
PyObject * (15)

EXAMPLE FUNCTION INVOCATION

CALLER: foo(3, 5)

function code

0	LOAD_FAST	0 (x)
2	LOAD_FAST	1 (y)
4	BINARY_MULTIPLY	
6	LOAD_CONST	1 (17)
8	BINARY_ADD	
10	RETURN_VALUE	



co_varnames

PyObject * (3)	PyObject * (5)
----------------	----------------

co_consts

PyObject * (None)	PyObject * (17)
-------------------	-----------------

object stack

PyObject * (17)


PyObject * (15)

EXAMPLE FUNCTION INVOCATION

CALLLED: `foo(3, 5)`

function code

0	LOAD_FAST	0 (x)
2	LOAD_FAST	1 (y)
4	BINARY_MULTIPLY	
6	LOAD_CONST	1 (17)
8	BINARY_ADD	
10	RETURN_VALUE	



co_varnames

PyObject * (3)	PyObject * (5)
----------------	----------------

co_consts

PyObject * (None)	PyObject * (17)
-------------------	-----------------

object stack

PyObject * (17)

POP!




PyObject * (15)

EXAMPLE FUNCTION INVOCATION

CALLLED: foo(3, 5)

function code

0	LOAD_FAST	0 (x)
2	LOAD_FAST	1 (y)
4	BINARY_MULTIPLY	
6	LOAD_CONST	1 (17)
8	BINARY_ADD	
10	RETURN_VALUE	



co_varnames

PyObject * (3)	PyObject * (5)
----------------	----------------

co_consts

PyObject * (None)	PyObject * (17)
-------------------	-----------------

object stack

PyObject * (17)

PyObject * (15)

POP!




EXAMPLE FUNCTION INVOCATION

CALLLED: foo(3, 5)

function code

0	LOAD_FAST	0 (x)
2	LOAD_FAST	1 (y)
4	BINARY_MULTIPLY	
6	LOAD_CONST	1 (17)
8	BINARY_ADD	
10	RETURN_VALUE	



co_varnames

PyObject * (3)	PyObject * (5)
----------------	----------------

co_consts

PyObject * (None)	PyObject * (17)
-------------------	-----------------

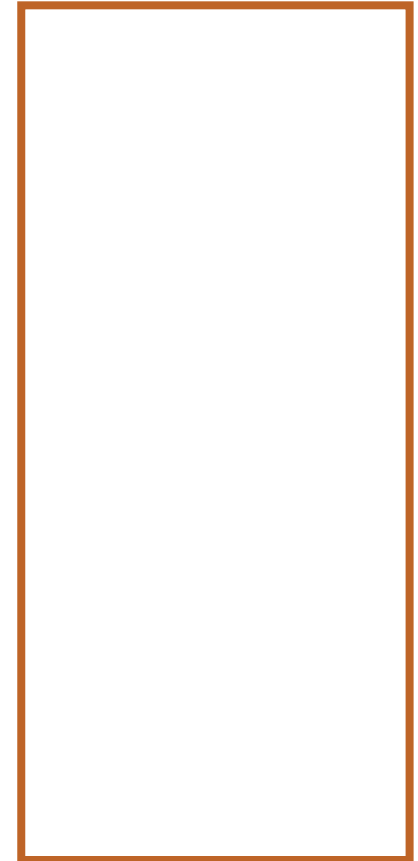
object stack

__add__

PyObject * (17)
PyObject * (15)



PyObject * (32)




EXAMPLE FUNCTION INVOCATION

CALLLED: `foo(3, 5)`

function code

0	LOAD_FAST	0 (x)
2	LOAD_FAST	1 (y)
4	BINARY_MULTIPLY	
6	LOAD_CONST	1 (17)
8	BINARY_ADD	
10	RETURN_VALUE	



co_varnames

PyObject * (3)	PyObject * (5)
----------------	----------------

co_consts

PyObject * (None)	PyObject * (17)
-------------------	-----------------

object stack

PyObject * (32)

PyObject * (32)


PUSH!

EXAMPLE FUNCTION INVOCATION

CALLLED: foo(3, 5)

function code

0	LOAD_FAST	0 (x)
2	LOAD_FAST	1 (y)
4	BINARY_MULTIPLY	
6	LOAD_CONST	1 (17)
8	BINARY_ADD	
10	RETURN_VALUE	



co_varnames

PyObject * (3)	PyObject * (5)
----------------	----------------

co_consts

PyObject * (None)	PyObject * (17)
-------------------	-----------------

object stack


PyObject * (32)

EXAMPLE FUNCTION INVOCATION

CALLLED: foo(3, 5)

function code

0	LOAD_FAST	0 (x)
2	LOAD_FAST	1 (y)
4	BINARY_MULTIPLY	
6	LOAD_CONST	1 (17)
8	BINARY_ADD	
10	RETURN_VALUE	



co_varnames

PyObject * (3)	PyObject * (5)
----------------	----------------

co_consts

PyObject * (None)	PyObject * (17)
-------------------	-----------------

object stack

PyObject * (32)

POP!

PyObject * (32)




EXAMPLE FUNCTION INVOCATION

CALLLED: `foo(3, 5)`

function code

0	LOAD_FAST	0 (x)
2	LOAD_FAST	1 (y)
4	BINARY_MULTIPLY	
6	LOAD_CONST	1 (17)
8	BINARY_ADD	
10	RETURN_VALUE	



co_varnames

PyObject * (3)	PyObject * (5)
----------------	----------------

co_consts

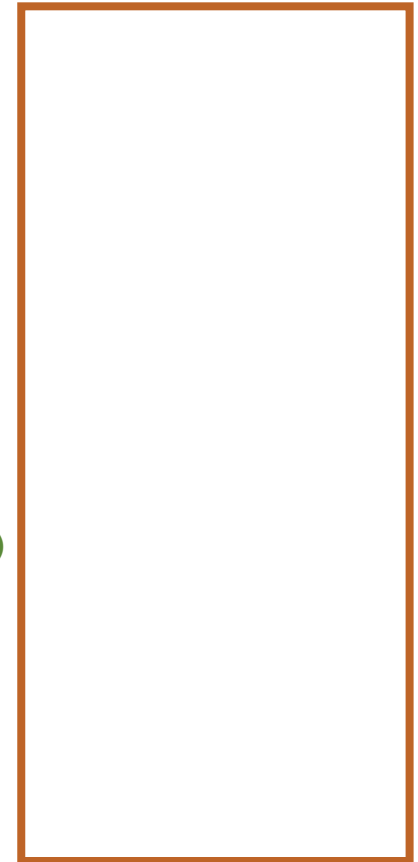
PyObject * (None)	PyObject * (17)
-------------------	-----------------

(return to caller)

PyObject * (32)



object stack



Parallelism in Python

- Take some LU-decompositon and inversion example code written in pure python
- Run it serially over a set of random matrices
- Try improving performance by multithreading
- Try improving performance by multiprocessing

Example code in the repo if you (or I) fall behind!

About Concurrency

- Parallelism: Do multiple things at the same time
- Concurrency: Deal with multiple things going on at the same time

Python supports multiple paradigms for achieving concurrent programming including os-level threads and also native coroutines

- Parallelism: Do multiple things at the same time
- Concurrency: Deal with multiple things going on at the same time
- Cooking example

Python supports multiple paradigms for achieving concurrent programming including os-level threads and also native coroutines

PYTHON COROUTINE EXAMPLE

```
import asyncio
```

```
import aiohttp
```

```
BASE_URL = 'https://ericappelt.com/animals/'
```

```
async def speak(animal, session):
```

```
    async with session.get('{}{}'.format(BASE_URL, animal)) as response:  
        response.raise_for_status()  
        sound = await response.text()
```

```
    return 'The {} says "{}".'.format(animal, sound)
```

Day Two

- Packaging, pip, and anaconda
- Numpy
- Revisiting matrix inversions with numpy
- A quick tour of scientific packages
- Jupyter notebooks and literate programming
- Making pretty and interactive plots

Repo: <https://github.com/appeltel/python-hpc-seminar>

(repo has links, code examples, etc...)

External Libraries

- pip: download and additional python libraries
- pypi/warehouse: central repository for community libraries
- virtualenv: tool for managing lightweight python “environments” of installed libraries
- system python: Python installed as part of your operating system - you don't want to mess with this

Quick Demo (... or read below if something goes wrong ...)

```
eric@laptop:tmp$ python -m virtualenv venv
Using base prefix '/Users/eric/.pyenv/versions/3.6.2'
New python executable in /Users/eric/tmp/venv/bin/python
Installing setuptools, pip, wheel...done.
eric@laptop:tmp$ source venv/bin/activate
(venv) eric@laptop:tmp$ pip install numpy
Collecting numpy
  Downloading numpy-1.14.2-cp36-cp36m-macosx_10_6_intel.macosx_10_9_intel.ma
ntel.macosx_10_10_x86_64.whl (4.7MB)
    100% |██████████████████████████████████████████████████████████████████| 4.7MB 193kB/s
Installing collected packages: numpy
Successfully installed numpy-1.14.2
(venv) eric@laptop:tmp$ python -c "import numpy as np; print(np.arange(5))"
[0 1 2 3 4]
(venv) eric@laptop:tmp$ deactivate
eric@laptop:tmp$ rm -r venv
eric@laptop:tmp$ █
```



- Pre-packaged distribution of python and many scientific libraries and tools
- Tools for managing environments, installing yet more packages
- Available on ACCRE cluster via LMOD

Intro to Numpy



NumPy is the fundamental package for scientific computing with Python. It contains among other things:

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code
- useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

Quickstart:

<https://docs.scipy.org/doc/numpy-dev/user/quickstart.html>

- Some experimentation on the REPL
- Revisiting the matrix inversion code
- Look at multithreading performance

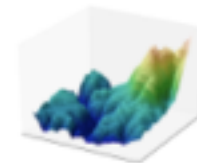
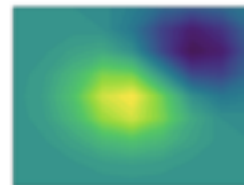
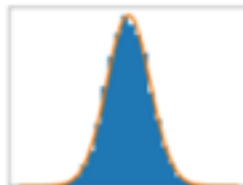
Tour of Libraries



- The SciPy library provides user friendly numerical integration and optimization functions
- Built on NumPy
- SciPy.org is the organization maintaining numpy, scipy, and a set of related tools
- These tools are often referred to as the SciPy stack

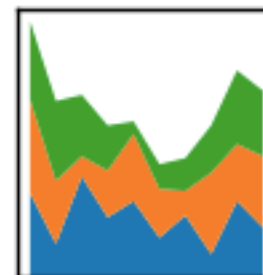
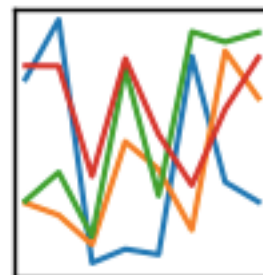
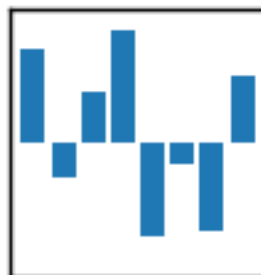
matplotlib

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and **IPython** shells, the **Jupyter** notebook, web application servers, and four graphical user interface toolkits.



pandas

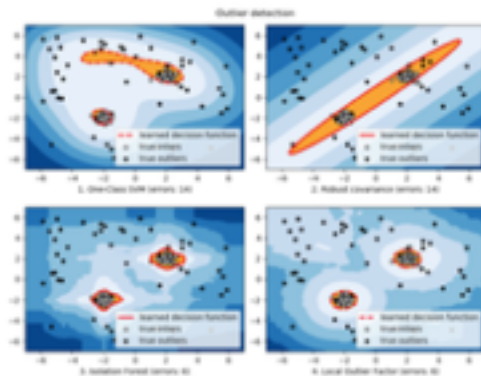
$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$



pandas is a **Python** package providing fast, flexible, and expressive data structures designed to make working with "relational" or "labeled" data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, **real world** data analysis in Python. Additionally, it has the broader goal of becoming **the most powerful and flexible open source data analysis / manipulation tool available in any language**. It is already well on its way toward this goal.

pandas is well suited for many different kinds of data:

- Tabular data with heterogeneously-typed columns, as in an SQL table or Excel spreadsheet
- Ordered and unordered (not necessarily fixed-frequency) time series data.
- Arbitrary matrix data (homogeneously typed or heterogeneous) with row and column labels
- Any other form of observational / statistical data sets. The data actually need not be labeled at all to be placed into a pandas data structure

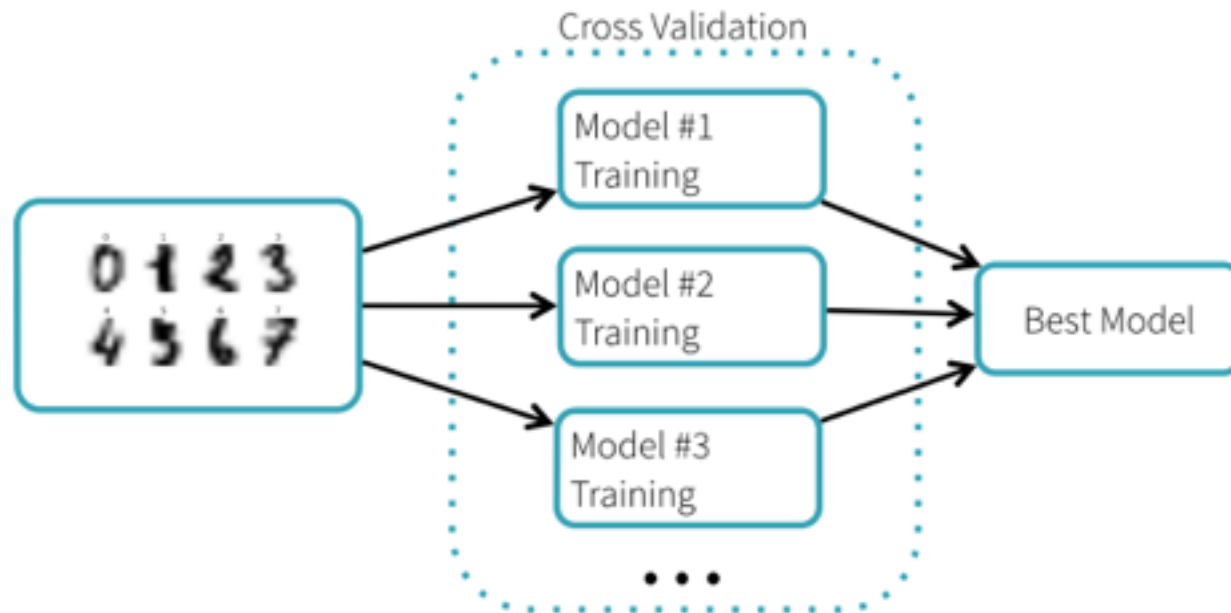


scikit-learn

Machine Learning in Python

- Simple and efficient tools for data mining and data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

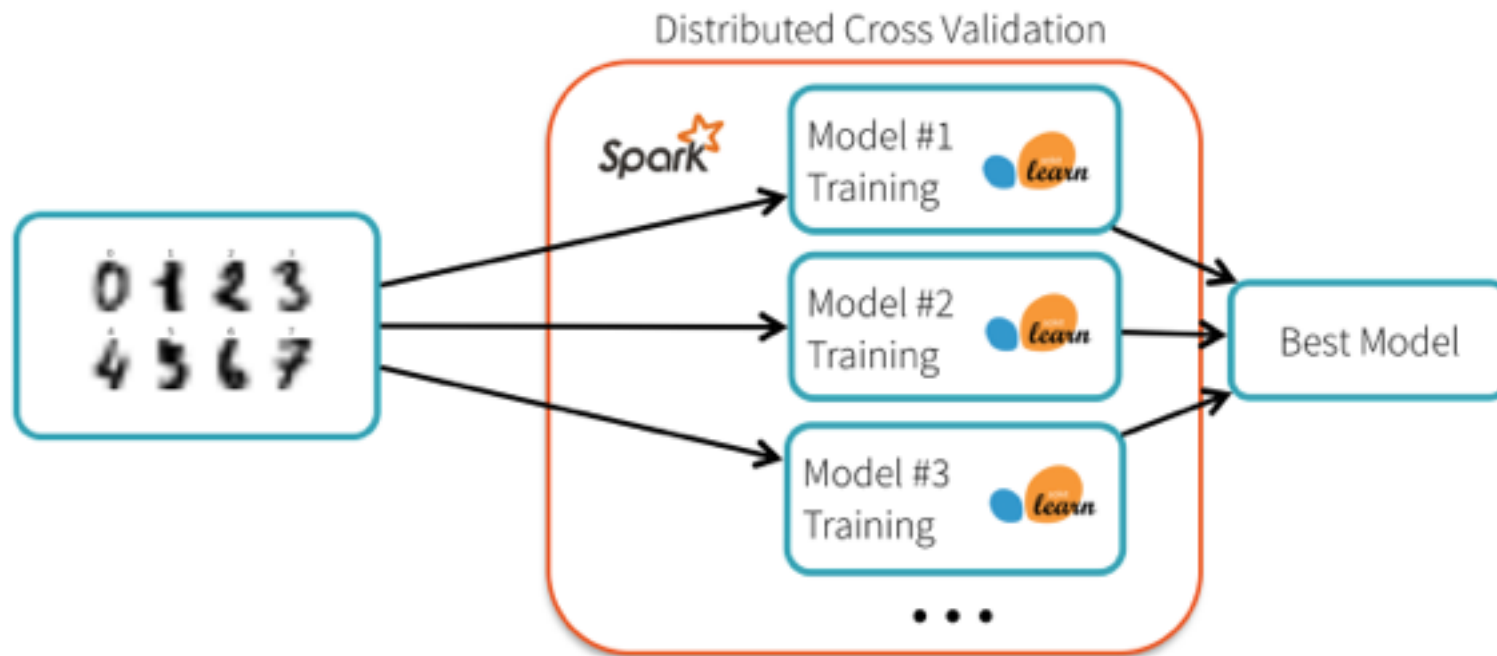
Training a random forest classifier to recognize handwritten digits:



Good candidate for parallelization and distribution over multiple cluster nodes, see:

<https://databricks.com/blog/2016/02/08/auto-scaling-scikit-learn-with-apache-spark.html>

Easy to do on a spark cluster:



- Can it also be done within a slurm job that sets up a joblib cluster for the training?
- Can we use our big data cluster for this?
- How is ACCRE performance?

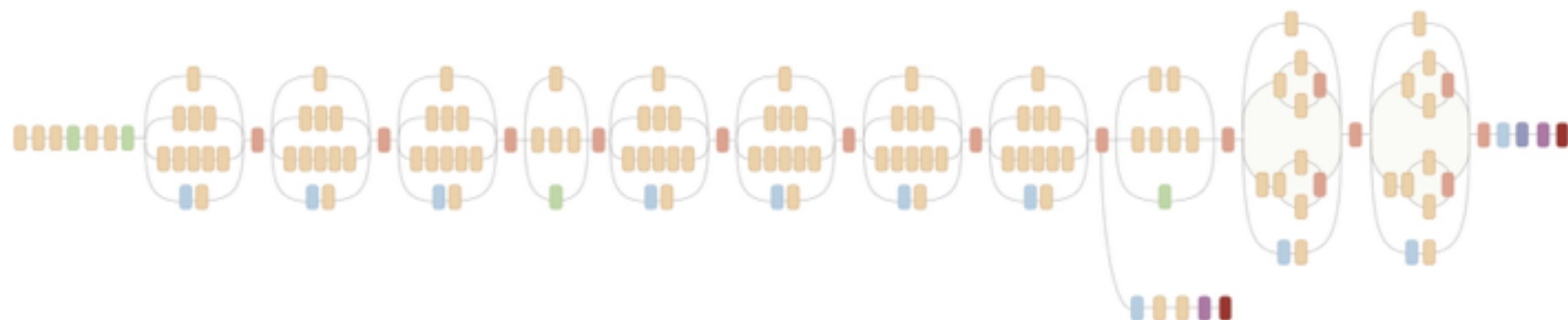


- Numerical computation with data flow graphs
- Edges represent mathematical operations
- Vertices represent tensors (arrays, matrices)
- Common API for deployment to CPU/GPU
- Deep Learning applications

Fun demo of deep learning: playground.tensorflow.org

TENSORFLOW - IMAGE RECOGNITION

Inception-v3 model:



- Convolution
- AvgPool
- MaxPool
- Concat
- Dropout
- Fully connected
- Softmax



Jupyter Notebooks



- Interactive web application
- Embed live code, narrative, and visualizations into notebook files
- Can be run from the ACCRE cluster
- Not just python!



- Do some demos
- You can run these directly out of the repo:
- <https://github.com/appeltel/python-hpc-seminar>