An Introduction to Python at NERSC

NERSC Data Day 2016

Rollin Thomas
Data & Analytics Services Group
2016-08-22
20 Questions! At NERSC...

1. Do you use Python?
2. Do you use Python 3 (yet)?
3. Do you use Anaconda Python?
4. Have you ever used numpy/scipy?
5. ... multiprocessing?
6. ... mpi4py?
7. ... IPython/Jupyter?
8. ... let’s make it 8 questions.
Python is Popular

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>-</td>
<td>Java</td>
<td>19.010%</td>
<td>-0.28%</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>-</td>
<td>C</td>
<td>11.303%</td>
<td>-3.43%</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>-</td>
<td>C++</td>
<td>5.800%</td>
<td>-1.94%</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>-</td>
<td>C#</td>
<td>4.907%</td>
<td>+0.07%</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td></td>
<td>Python</td>
<td>4.404%</td>
<td>+0.34%</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>▲</td>
<td>PHP</td>
<td>3.173%</td>
<td>+0.44%</td>
</tr>
<tr>
<td>7</td>
<td>9</td>
<td>▲</td>
<td>JavaScript</td>
<td>2.705%</td>
<td>+0.54%</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>▲</td>
<td>Visual Basic .NET</td>
<td>2.518%</td>
<td>-0.19%</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>▲</td>
<td>Perl</td>
<td>2.511%</td>
<td>+0.39%</td>
</tr>
<tr>
<td>10</td>
<td>12</td>
<td>▲</td>
<td>Assembly language</td>
<td>2.364%</td>
<td>+0.60%</td>
</tr>
</tbody>
</table>

www.tiobe.com/tiobe-index

codeval.com

For anyone interested in research and big data analysis, Python can be a powerful language to start with. Python has an engaged community and is updated often, with a new version released each year or so. According to Google Trends, learning Python is expected to become more and more popular.

www.bestprogramminglanguagefor.me
Why Python?

Clean, clear syntax makes it very easy to learn.

Multi-paradigm interpreted language.

Extremely popular language for teaching beginners...

… but stays useful beyond the beginner phase of programming:

Powerful data structures and constructs built into the language and standard libraries. Leveraging of C/C++/Fortran.

Huge collection of useful open source packages to re-use and extend.
Supporting Python is no longer optional at HPC centers like NERSC.

Maximizing Python performance on systems like Cori and Edison can be challenging:

- Interpreted, dynamic languages are harder to optimize.
- Python’s global interpreter lock is an issue for thread-level parallelism.
- Language design and implementation choices made without considering an HPC environment.

At the same time, users want NERSC to provide a familiar and portable Python environment.
Environment modules:
Environment modules project:
http://modules.sourceforge.net/

Always* “module load python”
Don’t use /usr/bin/python.
Using #!/usr/bin/env python: OK!

What is there?
module avail python

* Unless you install your own Python somehow.
Python Installations at NERSC

“NERSC-Built” Python
- Python “base” module.
- Add-on modules as desired.
- Meta-module simplifies setup.

Anaconda Python
- “Distribution” for large-scale data analytics, and scientific computing.
- ~200 packages but there is also “miniconda” bare-bones starter.
- Simplified package management and deployment (conda tool).
- Monolithic module, some add-on modules (h5py-parallel).

https://docs.continuum.io/anaconda/
Python Modules on Edison

NERSC-built:

```
module load python/[2.7.9]
python_base/2.7.9
numpy/1.9.2
scipy/0.15.1
matplotlib/1.4.3
ipython/3.1.0
```

Anaconda:

```
module load python/2.7-anaconda
module load python/3.5-anaconda
```

Above are the only currently recommended Python modules for Edison.
Python Modules on Cori

NERSC-built:

There aren’t any.

Anaconda:

```
module load python[2.7-anaconda]
module load python/3.5-anaconda
```

Above are the only currently recommended Python modules for Cori.
Anaconda Environment under Modules:

```
module load python/2.7-anaconda
conda create -p $PREFIX numpy...
conda create -n myenv numpy...
(won’t work for users without .condarc defining “envs_dirs”)
conda install basemap yt...
```

Your own Anaconda or Miniconda installation:

```
module unload python
wget https://repo.continuum.io/miniconda/Miniconda2-latest-Linux-x86_64.sh
/bin/bash Miniconda2-latest-Linux-x86_64.sh -b -p $PREFIX
export PATH=$PREFIX/bin:$PATH
conda install basemap yt...
```

Tips:
- Conda environments do **not** mix with virtualenv.
- Several ML environments via Anaconda at NERSC.
Anaconda Python provides access to Intel Math Kernel Library (MKL) for free:

- numpy
- scipy
- scikit-learn
- numexpr

MKL Service functions*:

```python
>>> import mkl
>>> mkl.get_max_threads()
2
>>> mkl.set_num_threads(1)
>>> mkl.get_max_threads()
1
```

*https://github.com/ContinuumIO/mkl-service
Intel Distribution for Python 2017 Beta

Available through Anaconda as well:

```bash
conda create -p $SCRATCH/idp \
-c intel intelpython2_core python=2
source activate $SCRATCH/idp
```

Features:

Leveraging Intel MKL, MPI, TBB, DAAL.

Intel-specific enhancements (FFT, threaded RNG, etc.).
Multi-Node Parallelism: mpi4py

MPI support via mpi4py (2.0.0)
Added earlier this year.
Includes MPI-3 features.

Compiled against Cray libraries.
Built into Anaconda modules on Edison and Cori.

Non-Anaconda route:
module load mpi4py

DIY mpi4py builders… see me.
Python’s “import” statement is file metadata intensive (.py, .pyc, .so open/stat calls).
- Becomes more severe as the number of Python processes trying to access files increases.
- Result: Very slow times to just start Python applications at larger concurrency (MPI).

**BEST POSSIBLE PERFORMANCE IS SHIFTER:**
- Eliminates metadata calls off the compute nodes.
- Paths to .so libraries can be cached via ldconfig.

Other approaches:
- Pack up software to compute nodes ([python-mpi-bcast](https://github.com/nerc-python-mpi-bcast)).
- Install software to $SCRATCH or /global/common.
Multiprocessing and Process Spawning

You can use multiprocessing for on-node throughput jobs.

Combining multiprocessing with mpi4py, mixed results.

Combining mpi4py and subprocess?
Works to spawn serial, compiled executables.
Just don’t compile those with Cray wrappers cc, CC, ftn.
Do `module load gcc` and use gcc, g++, gfortran.
Jupyter at NERSC and on Cori


New way to interact with NERSC HPC resources:
Old: Use ssh or NX to get to command line.
New: Open a notebook, create a narrative.

Move to Cori:
- Access to $SCRATCH.
- Integration with SLURM.
- Eventually Burst Buffer.
- New ways of using Cori.
  - DASK, PySpark, IJulia...
Live Demo
### SLURM Magic Commands

#### In [1]:
```
squeue -u rthomas
```

#### Out[1]:

<table>
<thead>
<tr>
<th>JOBID</th>
<th>USER</th>
<th>ACCOUNT</th>
<th>NAME</th>
<th>PARTITION</th>
<th>QOS</th>
<th>NODES</th>
<th>TIME_LIMIT</th>
<th>TIME</th>
<th>ST</th>
<th>PRIORITY</th>
<th>SUBMIT_TIME</th>
<th>START_TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2875563</td>
<td>rthomas</td>
<td>mpccc</td>
<td>try.sh</td>
<td>regular</td>
<td>normal_reg</td>
<td>150</td>
<td>1:00</td>
<td>0:00</td>
<td>RD 64832</td>
<td>2016-06-19T15:33:09</td>
<td>NaN</td>
</tr>
</tbody>
</table>

#### In [2]:
```
sprio -j 2875563
```

#### Out[2]:

<table>
<thead>
<tr>
<th>JOBID</th>
<th>PRIORITY</th>
<th>AGE</th>
<th>FAIRSHARE</th>
<th>QOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>64832</td>
<td>32</td>
<td>0</td>
<td>64800</td>
</tr>
</tbody>
</table>

#### In [3]:
```
sacct -u rthomas
```

#### Out[3]:

<table>
<thead>
<tr>
<th>JobID</th>
<th>JobName</th>
<th>Partition</th>
<th>Account</th>
<th>AllocCPUS</th>
<th>State</th>
<th>ExitCode</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>--------</td>
<td>-----------</td>
<td>---------</td>
<td>-----------</td>
<td>-----------</td>
<td>----------</td>
</tr>
<tr>
<td>1</td>
<td>try.sh</td>
<td>debug</td>
<td>mpccc</td>
<td>64</td>
<td>FAILED</td>
<td>1:0</td>
</tr>
<tr>
<td>2</td>
<td>batch</td>
<td>mpccc</td>
<td>64</td>
<td>FAILED</td>
<td>1:0</td>
<td>NaN</td>
</tr>
<tr>
<td>3</td>
<td>extern</td>
<td>mpccc</td>
<td>64</td>
<td>COMPLETED</td>
<td>0:0</td>
<td>NaN</td>
</tr>
<tr>
<td>4</td>
<td>shifter</td>
<td>mpccc</td>
<td>32</td>
<td>FAILED</td>
<td>1:0</td>
<td>NaN</td>
</tr>
<tr>
<td>5</td>
<td>try.sh</td>
<td>debug</td>
<td>mpccc</td>
<td>64</td>
<td>FAILED</td>
<td>1:0</td>
</tr>
<tr>
<td>6</td>
<td>batch</td>
<td>mpccc</td>
<td>64</td>
<td>FAILED</td>
<td>1:0</td>
<td>NaN</td>
</tr>
</tbody>
</table>
Python on Cori Phase II

**Knights Landing (KNL)**
- 2x cores per node
- Slower clock rate
- Less memory/core.

**Single-thread or flat MPI**
Python won’t be great.

**Advice:**
- Leverage threaded, vectorized math/specialized libraries.
- Consider writing Cython/C extensions you can vectorize?
- Learn about Intel Python and Intel profiling tools.
Conclusion

Python is an integral element of NERSC’s Data Intensive Science portfolio.

We want users to have a:

- **familiar** Python environment
- **productive** Python experience
- **performant** Python software stack

Pursuing new ways to empower Python & data users.

Always looking for feedback, advice, and even help:
- consult@nersc.gov
- https://help.nersc.gov
- rctthomas@lbl.gov
National Energy Research Scientific Computing Center