Python on Frontier

Michael Sandoval

HPC Engineer - User Assistance Group
Oak Ridge Leadership Computing Facility (OLCF)
Oak Ridge National Laboratory (ORNL)

February 28, 2024

Overview

- What to Expect on Frontier
- Virtual Environments
  - What are they and how do they work?
  - Options on Frontier
    - Cray-python
    - Miniforge3 (new Conda module!!)
- Using `cray-python`
- Using Miniforge
- General Best Practices

Moving to Frontier: What to Expect

The big takeaway….

- No more Power architecture, x86 is back!
  - Easier to install from pre-compiled binaries
  - Source installs are “easier”
  - Plays nice with conda/mamba and pip
- Should play nicer with Slurm (see: Andes)
- GPU workflow is now the biggest hurdle with the switch to AMD
  - Won’t be talking about this today
- New conda module! (Miniforge)

Virtual Environments

- What are they?
  - Isolated directory trees that help you manage various packages or different versions of Python
- Why are they beneficial?
  - Dependencies of one package might clash with dependencies of another package
  - Allows you to install new packages without modifying a "base" installation
  - Unique environments can be used on a per-project basis
- We will only be discussing Python3 approaches

Managing Python gets complicated sometimes….

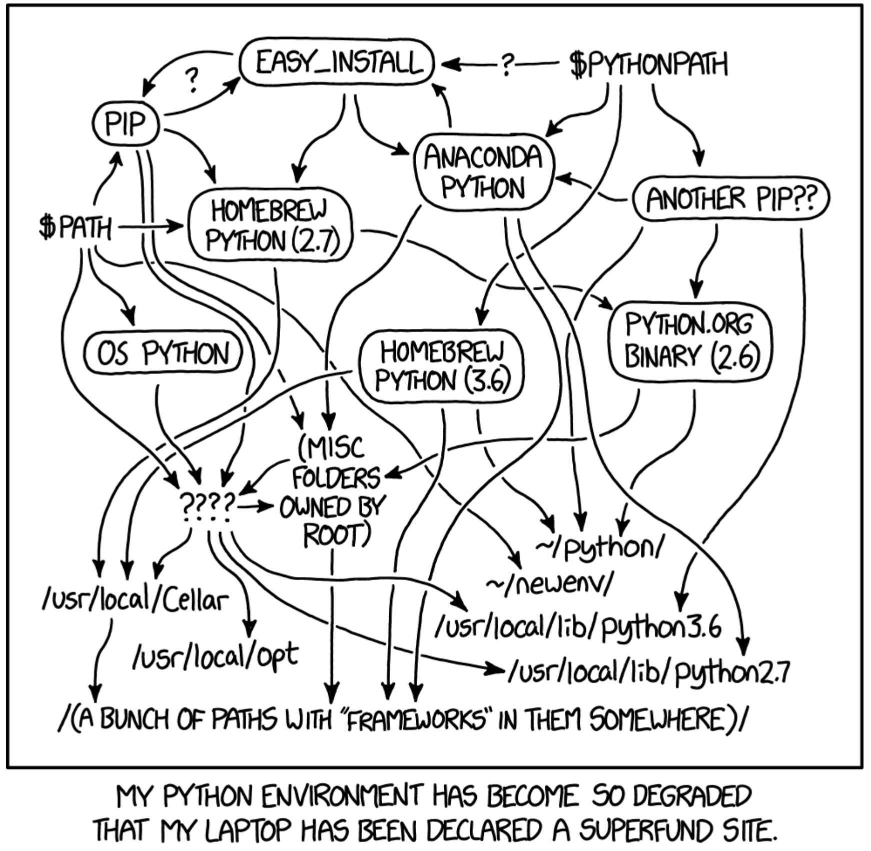


Image credit: https://xkcd.com/1987/

What about on Frontier?

Two main options:

1. Use the `cray-python` module
   - Supports **`venv`** syntax
   - Comes with pre-installed packages like numpy, scipy, mpi4py tuned for Cray machines

2. Use the `python/3.10-miniforge3` module
   - https://github.com/conda-forge/miniforge
   - Essentially just miniconda, but uses conda-forge by default
   - Supports **`conda`** and **`mamba`** syntax
   - Similar workflow to what is used on Andes, Summit
   - Summit now has this module (Andes in unchanged)

Comparing the options on Frontier

- `cray-python` module
  - Pros:
    - Pre-installed libraries tuned for Cray machines
  - Cons:
    - Extremely minimal
    - Highly dependent on pip
    - Restricted to version of module
    - Can't switch between different Python versions that easily using `venv`

Personally, I highly recommend using the Miniforge module instead of cray-python because you can do all of what cray-python can do and more using Miniforge (even if you don't end up using it for conda and just use pip)

- Miniforge (Conda/Mamba)
  - Pros:
    - Let's you manage multiple Python versions, not just environments
    - "Easy" to install dependencies based on your current environment
    - Highly customizable
    - Similar workflow across other OLCF systems
    - Easy to move to the burst buffer
  - Cons:
    - Highly dependent on pre-compiled binaries **(but can still use pip)**
    - Not specifically tuned for Cray machines unless you build from source

The `cray-python` Module

- Loading the modules: `module load cray-python`
- Create virtual environments by doing:
  `python3 -m venv /path/to/my_env`
- This creates a set of directories at the specified location, which will contain everything unique to that virtual environment
- How to activate and deactivate the environment:
  - From the command line: `source /path/to/my_env/bin/activate`
  - From the command line: `deactivate`
  - Using a shebang line : `#!/path/to/my_env/bin/python3`
- After activating, you can then install new packages using pip

The Miniforge Module

- Miniforge is just miniconda but targets the conda-forge channel by default
  - Bonus: has mamba pre-installed
  - Does *not* come with pre-installed packes like NumPy, Scipy, etc. in the base environment
- Loading the module: `module load python/3.10-miniforge3`
- Create Environments:

```
conda create -n my_env python=…
conda create -p /path/to/my_env python=…
```

- Activate/Deactivate Environments:

```
source activate /path/to/my_env
source deactivate
```

- Install packages (can use default channel or explicit channel name):

`conda install package_name` **or** `conda install -c conda-forge package_name`

More Conda Info

- See our Conda Basics guide with a quick-reference list here: https://docs.olcf.ornl.gov/software/python/conda_basics.html#conda-quick
- Conda's official user guide: https://docs.conda.io/projects/conda/en/latest/user-guide/index.html

Environment Locations

- Most default environment locations are at `$HOME` on NFS, but those typically initialize and run *much* slower on Frontier
- Environments stored / created on Orion perform faster than NFS, but still are relatively slow to initialize for complex environments and when running at scale
- Use the burst buffer (NVMe) instead!
  - https://docs.olcf.ornl.gov/software/python/sbcast_conda.html
  - Not necessary for everyone, but highly recommended for large node counts or users running into performance problems
  - If you use PyTorch, may consider this
- TLDR: **NVMe > Orion >> NFS**
- For collaboration, you can use Project shared areas like: `/ccs/proj/<proj_id>` or `/lustre/orion/proj-shared`
- Although faster, be careful storing things in `$MEMBERWORK` or `$PROJWORK` (Orion) because it might get purged.

Best Practices - I

- Make note of your pip cache location by running: `pip cache info`
  - May need to clean it from time-to-time with: `pip cache purge`
- Similarly, clean your `conda` cache occasionally: `conda clean –a`
- Explicitly use "`python3`" (or "`python2`") instead of the "`python`" alias
  - Better yet, explicitly put the full path to your Python installation
- In general, most python packages assume use of GCC
  - Recommended to use PrgEnv-gnu , especially when building from source

Best Practices - II

- Deactivate virtual environments first before switching PrgEnv modules
- Deactivate virtual environments before entering batch/interactive jobs
  - Some deactivation syntax won't work properly if entering a job already activated
  - Always better to enter any form of job with a fresh login shell and module environment
- When submitting a batch job that uses virtual environments, it's good to submit like so:

```
sbatch --export=NONE submit.sl
```

This means you'll have to activate all your modules / your virtual env in the batch script

Must use this at top of batch script if using –export=NONE: unset SLURM_EXPORT_ENV

Best Practices - III

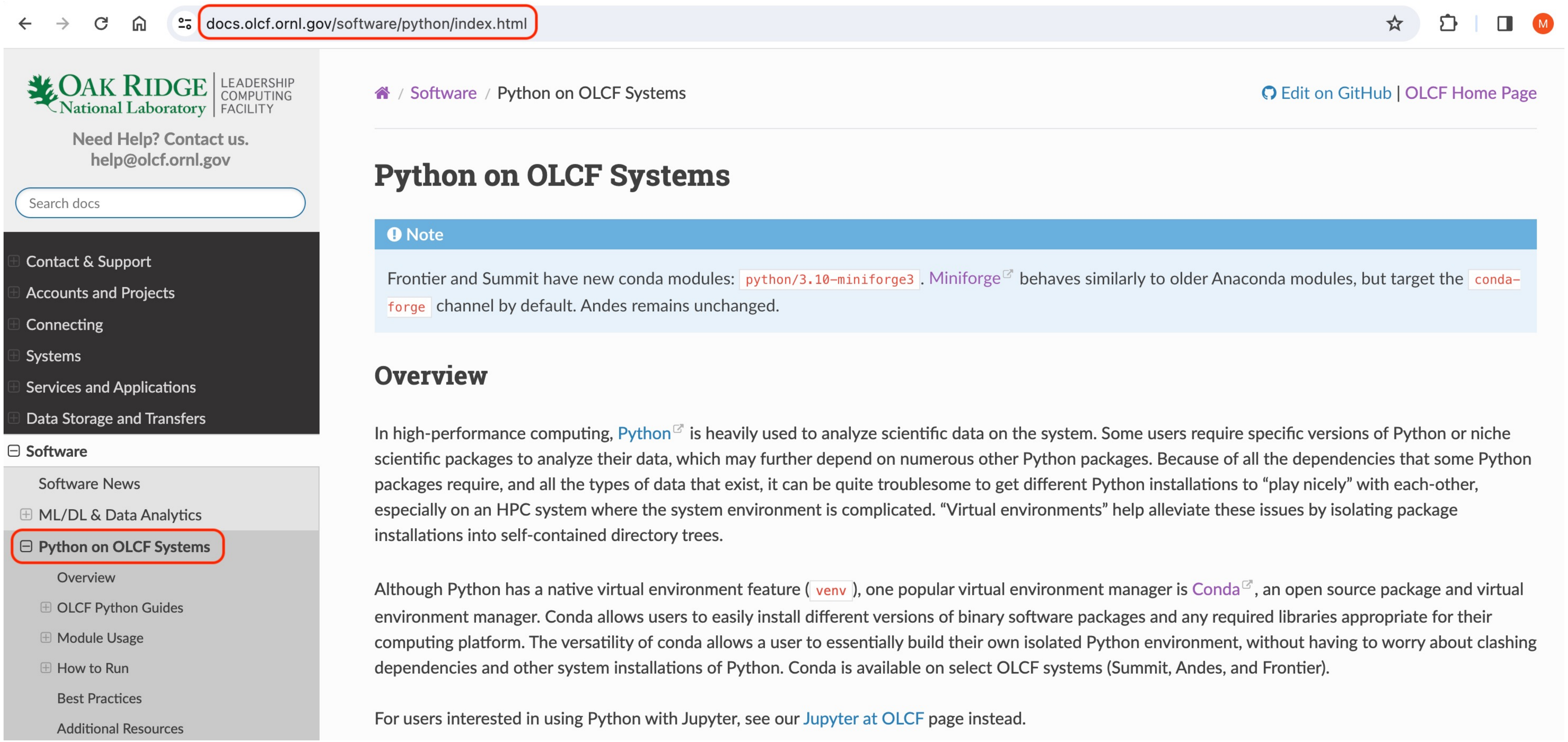
- To “export”/“import” your current environment:
  - For `venv`:

```
python3 –m pip freeze > requirements.txt
python3 –m pip install –r requirements.txt
```

  - For `conda`:

```
conda env export > environment.yml
conda env create –f environment.yml
```

- For unbuffered input, either enable that setting w/ SLURM or use the `-u` Python flag like so: `python3 –u script.py`



Python on OLCF Systems

Note

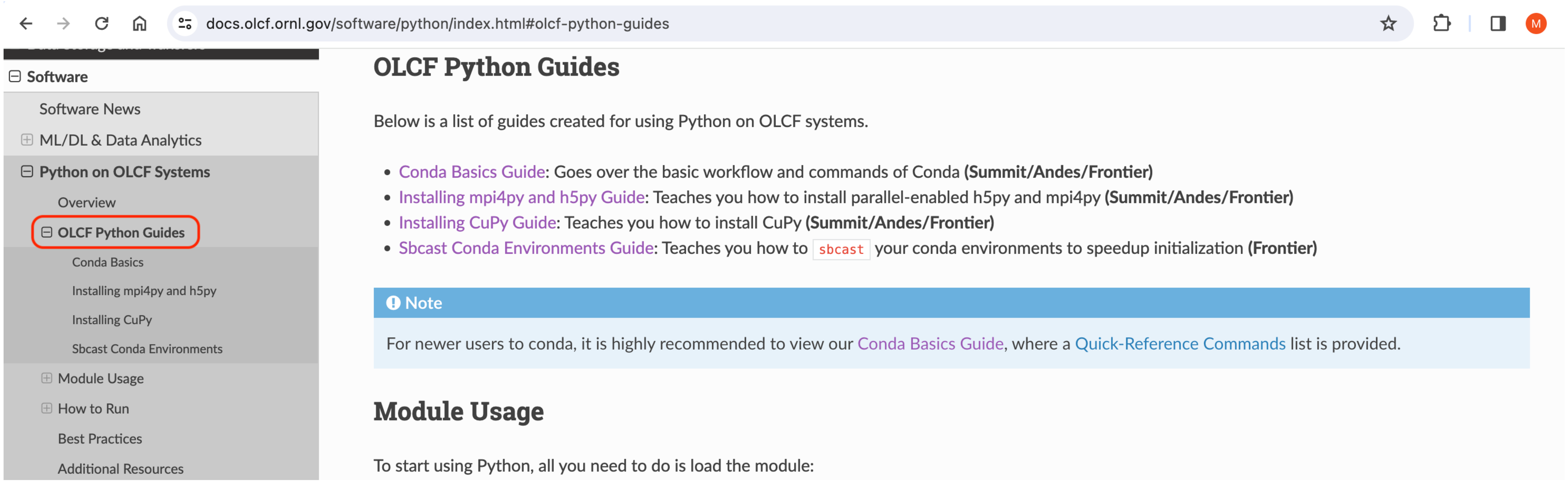
Frontier and Summit have new conda modules: `python/3.10-miniforge3`. Miniforge behaves similarly to older Anaconda modules, but target the `conda-forge` channel by default. Andes remains unchanged.

Overview

In high-performance computing, Python is heavily used to analyze scientific data on the system. Some users require specific versions of Python or niche scientific packages to analyze their data, which may further depend on numerous other Python packages. Because of all the dependencies that some Python packages require, and all the types of data that exist, it can be quite troublesome to get different Python installations to "play nicely" with each-other, especially on an HPC system where the system environment is complicated. "Virtual environments" help alleviate these issues by isolating package installations into self-contained directory trees.

Although Python has a native virtual environment feature (`venv`), one popular virtual environment manager is Conda, an open source package and virtual environment manager. Conda allows users to easily install different versions of binary software packages and any required libraries appropriate for their computing platform. The versatility of conda allows a user to essentially build their own isolated Python environment, without having to worry about clashing dependencies and other system installations of Python. Conda is available on select OLCF systems (Summit, Andes, and Frontier).

For users interested in using Python with Jupyter, see our Jupyter at OLCF page instead.



OLCF Python Guides

Below is a list of guides created for using Python on OLCF systems.

- Conda Basics Guide: Goes over the basic workflow and commands of Conda **(Summit/Andes/Frontier)**
- Installing mpi4py and h5py Guide: Teaches you how to install parallel-enabled h5py and mpi4py **(Summit/Andes/Frontier)**
- Installing CuPy Guide: Teaches you how to install CuPy **(Summit/Andes/Frontier)**
- Sbcast Conda Environments Guide: Teaches you how to `sbcast` your conda environments to speedup initialization **(Frontier)**

Note

For newer users to conda, it is highly recommended to view our Conda Basics Guide, where a Quick-Reference Commands list is provided.

Module Usage

To start using Python, all you need to do is load the module:

Additional Resources

- PyTorch on Frontier guide/seminar coming soon!
  - https://www.olcf.ornl.gov/calendar/pytorch-on-frontier/
- Our OLCF Python docs: https://docs.olcf.ornl.gov/software/python/index.html
- Official Python docs: https://docs.python.org/3/library/venv.html
- Official Conda docs: https://docs.conda.io/projects/conda/en/latest/user-guide/index.html
- Submit a ticket to help@olcf.ornl.gov
- Questions?