PYTHON USAGE BASICS: TETRALITH

LEARNING OBJECTIVES

- System Python
- Available Python modules
- Checking availability of packages
- Controlling the Python version within your scripts
- NSC documentation

SYSTEM PYTHON

```
mylaptop$ ssh tetralith.nsc.liu.se

$ which python
/usr/bin/python
$ python -V
Python 2.7.5
$
$ which python3
/usr/bin/python3
$ python3 -V
Python 3.6.8
$
```

Questions

- When should you use the system python?
- Thinlinc system python?

PYTHON MODULES (1)

```
$ module avail Python/

------------------------------------------------------------------------ /software/sse/modules -----------------------
   Python/recommendation             (D)    Python/2.7.14-nsc1-intel-2018a-eb           Python/3.6.3-anaconda-5.0.1-nsc
   Python/2.7.14-anaconda-5.0.1-nsc1        Python/2.7.15-anaconda-5.3.0-extras-nsc1    Python/3.6.4-nsc1-intel-2018a-e
   Python/2.7.14-nsc1-gcc-2018a-eb          Python/2.7.15-env-nsc1-gcc-2018a-eb         Python/3.6.4-nsc2-intel-2018a-e

  Where:
   D:  Default Module

Use "module spider" to find all possible modules.
Use "module keyword key1 key2 ..." to search for all possible modules matching any of the "keys".
$
$ module load Python/3.7.0-anaconda-5.3.0-extras-nsc1
$ which python
/software/sse/easybuild/prefix/software/Anaconda3/5.3.0-extras-nsc1/bin-wrapped/python
$ python -V
Python 3.7.0
$
$ which python3
/usr/bin/python3
$ python3 -V
Python 3.6.8
$
```

Main distinctions:

- python 2.7.x / python 3.x.y
- anaconda / NSC builds
- intel / gcc

PYTHON MODULES (2)

Some guidelines to help you choose a module:

- There are generally more packages included in the Anaconda installations
- If there are modules that only differ in the -nsc build/installation tag, then choose the one with the highest integer (e.g. nsc2 rather than nsc1)

PYTHON MODULES (3)

Basic usage

```
$ module load Python/3.7.0-anaconda-5.3.0-extras-nsc1
$ python
Python 3.7.0 (default, Jun 28 2018, 13:15:42)
[GCC 7.2.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.

>>> import numpy
>>> numpy.linspace(0, 2, 9)
array([0.  , 0.25, 0.5 , 0.75, 1.  , 1.25, 1.5 , 1.75, 2.  ])
>>>
```

PYTHON MODULES (4)

Anaconda modules. Conda User Guide

Check available packages

```
$ module load Python/3.7.0-anaconda-5.3.0-extras-nsc1
$ conda list
# packages in environment at /software/sse/easybuild/prefix/software/Anaconda3/5.3.0-extras-nsc1:
#
# Name                    Version                   Build  Channel
_ipyw_jlab_nb_ext_conf    0.1.0                    py37_0
agate                     1.6.1                    py37_2
alabaster                 0.7.11                   py37_0
anaconda                  5.3.0                    py37_0
anaconda-client           1.7.2                    py37_0
anaconda-navigator        1.9.2                    py37_0
anaconda-project          0.8.2                    py37_0
...
$
$ conda list | grep -i scipy
scipy                     0.19.1           py36h9976243_3
```

PYTHON MODULES (5)

NSC build modules: Check available packages

```
$ module load Python/3.6.4-nsc2-intel-2018a-eb
$ pip list --format=legacy
asn1crypto (0.24.0)
bcrypt (3.1.4)
bitstring (3.1.5)
blist (1.3.6)
certifi (2018.8.24)
chardet (3.0.4)
...
$
$ pip list --format=legacy | grep -i scipy
scipy (1.0.0)
```

CONTROLLING PYTHON VERSION IN YOUR SCRIPTS(1)

Executing python scripts. Either

```
$ python py3-example-envpy.py
```

or

```
$ ./py3-example-envpy.py
```

Note - your script must have execute permissions and shebang line

```
$ ls -l py3-example-envpy.py
-rwxrwxr-x 1 struthers pg_nsc 127 Oct  1 11:49 py3-example.py
$
$ cat py3-example-envpy.py
#!/usr/bin/env python

# The following line is only compatable with Python 3.
assert 3 / 2 == 1.5

print('Script successful')
$
```

CONTROLLING PYTHON VERSION IN YOUR SCRIPTS(2)

The shebang line and the Python interpreter

Commonly, scripts hard coded to system python interpreter

```
#!/usr/bin/python
...
```

Can be set to use the environment python interpreter

```
#!/usr/bin/env python
...
```

Or hard coded to a module python interpreter

```
$ module load Python/3.7.0-anaconda-5.3.0-extras-nsc1
$ which python
/software/sse/easybuild/prefix/software/Anaconda3/5.3.0-extras-nsc1/bin-wrapped/python
$
$ cat my-python-script.py
#!/software/sse/easybuild/prefix/software/Anaconda3/5.3.0-extras-nsc1/bin-wrapped/python
...
```

CONTROLLING PYTHON VERSION IN YOUR SCRIPTS(3)

Examples: Script with no shebang line

```
$ cat py3-example-noshebang.py
# The following line is only compatable with Python 3.
assert 3 / 2 == 1.5

print('Script successful')
$
$ # system python
$ module purge
$ python py3-example-noshebang.py
Traceback (most recent call last):
  File "py3-example-no-shebang.py", line 2, in <module>
    assert 3 / 2 == 1.5
AssertionError
$ python3 py3-example-noshebang.py
Script successful
$ ./py3-example-noshebang.py
./py3-example-noshebang.py: line 2: assert: command not found
./py3-example-noshebang.py: line 4: syntax error near unexpected token `'Script successful''
./py3-example-noshebang.py: line 4: `print('Script successful')'
$
```

```
$
$ # module python (python v3 module)
$ module load Python/3.7.0-anaconda-5.3.0-extras-nsc1
$ python py3-example-noshebang.py
Script successful
$
$ # module python (python v2 module)
$ module load Python/2.7.15-anaconda-5.3.0-extras-nsc1
$ python py3-example-noshebang.py
Traceback (most recent call last):
  File "py3-example-noshebang.py", line 2, in <module>
    assert 3 / 2 == 1.5
AssertionError
$
```

CONTROLLING PYTHON VERSION IN YOUR SCRIPTS(4)

Examples: Script with shebang line (environment python)

```
$ cat py3-example-envpy.py
#!/usr/bin/env python

# The following line is only compatable with Python 3.
assert 3 / 2 == 1.5

print('Script successful')
$
$ # system python
$ module purge
$ python py3-example-envpy.py
Traceback (most recent call last):
  File "py3-example-envpy.py", line 2, in <module>
    assert 3 / 2 == 1.5
AssertionError
$ python3 py3-example-envpy.py
Script successful
$ ./py3-example-envpy.py
Traceback (most recent call last):
  File "./py3-example-envpy.py", line 4, in <module>
    assert 3 / 2 == 1.5
AssertionError
$
```

```
$
$ # module python (python v3 module)
$ module load Python/3.7.0-anaconda-5.3.0-extras-nsc1
$ python py3-example-envpy.py
Script successful
$ ./py3-example-envpy.py
Script successful
$
$ # module python (python v2 module)
$ module load Python/2.7.15-anaconda-5.3.0-extras-nsc1
$ python py3-example-envpy.py
Traceback (most recent call last):
  File "py3-example-envpy.py", line 4, in <module>
    assert 3 / 2 == 1.5
AssertionError
$ ./py3-example-envpy.py
Traceback (most recent call last):
  File "./py3-example-envpy.py", line 4, in <module>
    assert 3 / 2 == 1.5
AssertionError
$
```

CONTROLLING PYTHON VERSION IN YOUR SCRIPTS(5)

Examples: Script with shebang line (module python)

```
$ cat py3-example-modulepy.py
#!/software/sse/easybuild/prefix/software/Anaconda3/5.3.0-extras-nsc1/bin-wrapped/python

# The following line is only compatable with Python 3.
assert 3 / 2 == 1.5

print('Script successful')
$
$ # system python
$ module purge
$ python py3-example-modulepy.py
Traceback (most recent call last):
  File "py3-example-modulepy.py", line 2, in <module>
    assert 3 / 2 == 1.5
AssertionError
$ python3 py3-example-modulepy.py
Script successful
$ ./py3-example-modulepy.py
Script successful
$
```

```
$
$ # module python (python v3 module)
$ module load Python/3.7.0-anaconda-5.3.0-extras-nsc1
$ python py3-example-modulepy.py
Script successful
$ ./py3-example-modulepy.py
Script successful
$
$ # module python (python v2 module)
$ module load Python/2.7.15-anaconda-5.3.0-extras-nsc1
$ python py3-example-modulepy.py
Traceback (most recent call last):
  File "py3-example-modulepy.py", line 4, in <module>
    assert 3 / 2 == 1.5
AssertionError
$ ./py3-example-modulepy.py
Script successful
$
```

CONTROLLING PYTHON VERSION IN YOUR SCRIPTS(6)

SOME PERSONAL RECOMMENDATIONS - FOR REPRODUCIBLE, SCIENTIFIC CODES

If you do not need to install any external package(s): Try not to rely on the the environment to determine the python interpreter that will be used to execute your script, e.g.

- Include a shebang line and execute using *./my-script.py*
- Hard code a link to module python rather than the system python, e.g.

```
#!/software/sse/easybuild/prefix/software/Anaconda3/5.3.0-extras-nsc1/bin-wrapped/python
```

or, make sure the environment is clearly defined by e.g. wrapping the execution of you python script

```
$ cat wrap-script.sh
#!/bin/bash

module purge
module load Python/3.7.0-anaconda-5.3.0-extras-nsc1

python my-python-script.py

$
```

CONTROLLING PYTHON VERSION IN YOUR SCRIPTS(7)

SOME PERSONAL RECOMMENDATIONS - FOR REPRODUCIBLE, SCIENTIFIC CODES

If you need to install external package(s):

- Hard code a link to your own *virtualenv* or *conda* python (see next section)

CUSTOMIZING YOUR PYTHON ENVIRONMENT

LEARNING OBJECTIVES

- Introduction: package management, environment management
- Customizing your environment using conda
- Customizing your environment using virtual env
- When to use conda, when to use virtualenv

INTRODUCTION: PACKAGE MANAGEMENT

A package manager is a tool that automates the process of installing, updating, and removing packages (software).

- Pip is Pythons officially-sanctioned package manager, and is most commonly used to install packages published on the Python Package Index (PyPI)
- Conda is a general-purpose package management system, designed to build and manage software of any type from any language. As such, it also works well with Python packages.
- pip installs python packages within any environment; conda installs any package within conda environments.

PACKAGE MANAGEMENT: PIP

By default, third party packages installed using pip are typically placed in one of the directories pointed to by *site.getsitepackages*:

```
$ # system python
$ module purge
$ python
Python 2.7.5 (default, Aug  7 2019, 00:51:29)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-39)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import site
>>> site.getsitepackages()
['/usr/lib64/python2.7/site-packages', '/usr/lib/python2.7/site-packages', '/usr/lib/site-python']
>>> quit()
$
$ # NSC module python
$ module load Python/3.6.4-nsc2-intel-2018a-eb
$ python
Python 3.6.4 (default, Sep 19 2018, 00:23:31)
[GCC Intel(R) C++ gcc 6.4 mode] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import site
>>> site.getsitepackages()
['/software/sse/easybuild/prefix/software/Python/3.6.4-intel-2018a-nsc2/lib/python3.6/site-packages']
>>> quit()
$
```

```
$
$ # Anaconda module python
$ module load Python/3.7.0-anaconda-5.3.0-extras-nsc1
$ python
Python 3.7.0 (default, Jun 28 2018, 13:15:42)
[GCC 7.2.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import site
>>> site.getsitepackages()
['/software/sse/easybuild/prefix/software/Anaconda3/5.3.0-extras-nsc1/bin-wrapped/../lib/python3.7/site-packages']
>>> quit()
$
```

PACKAGE MANAGEMENT: CONDA

Default package installation

```
$ module load Python/3.6.3-anaconda-5.0.1-nsc1
$ conda env list
# conda environments:
#
base                  *  /software/sse/easybuild/prefix/software/Anaconda3/5.3.0-extras-nsc1

$ conda config --show
add_anaconda_token: True
add_pip_as_python_dependency: True
aggressive_update_packages:
  - ca-certificates
  - certifi
  - openssl
allow_non_channel_urls: False
...
download_only: False
envs_dirs:
  - /home/struthers/.conda/envs
  - /software/sse/easybuild/prefix/software/Anaconda3/5.3.0-extras-nsc1/envs
force: False
...
pinned_packages: []
pkgs_dirs:
  - /software/sse/easybuild/prefix/software/Anaconda3/5.3.0-extras-nsc1/pkgs
  - /home/struthers/.conda/pkgs
proxy_servers: {}
...
```

INTRODUCTION: (VIRTUAL) ENVIRONMENT MANAGEMENT

So on Tetralith it is not possible to install 3rd party python modules using pip or conda?

```
$ module load Python/3.6.4-nsc2-intel-2018a-eb
$ pip install "SomeProject"
...
$ module load Python/3.7.0-anaconda-5.3.0-extras-nsc1
$ conda install "SomeProject"
```

No - you can use e.g.

```
$ module load Python/3.6.4-nsc2-intel-2018a-eb
$ pip install "SomeProject" --user
```

BUT ...

INTRODUCTION: (VIRTUAL) ENVIRONMENT MANAGEMENT

Environment management is an important way of managing package dependencies.

- An environment consists of a certain python version and some packages.
- If you want to develop or use applications with different python or package version requirements, you need to set up different environments.
- If you want to share your python script(s)/workflows you should use environments
- In summary - you should use environments!

ENVIRONMENT MANAGEMENT(1): VIRTUALENV + PIP

Basic usage

```
$ module load Python/3.6.4-nsc2-intel-2018a-eb
$ virtualenv --system-site-packages myEnv
$ source myEnv/bin/activate
(myEnv) $ pip install "SomeProject"
(myEnv) $ ...
(myEnv) $ which python
~/myEnv/bin/python
(myEnv) $ deactivate
```

When you login next time, you do not have to create the environment again. Simply activate it using:

```
$ source myEnv/bin/activate
$ ...
```

ENVIRONMENT MANAGEMENT(2): VIRTUALENV + PIP

Simple example: astropy

```
$ cd myVirtualEnvs
$ module load Python/3.6.4-nsc2-intel-2018a-eb
$ virtualenv --system-site-packages Astropy-eg
$ source Astropy-eg/bin/activate
(Astropy-eg) $ pip install astropy
(Astropy-eg) $ pip install matplotlib
...
(astropy-eg) $ deactivate
$
```

ENVIRONMENT MANAGEMENT(3): CONDA

Basic usage

```
$ module load Python/3.7.0-anaconda-5.3.0-extras-nsc1
$ conda create --name myEnv
$ source activate myEnv
(myEnv) $ conda install "SomeProject"
(myEnv) $ ...
(myEnv) $ which python
~/.conda/envs/mvEnv/bin/python
(myEnv) $ source deactivate
```

When you login next time, you do not have to create the environment again. Simply activate it using:

```
$ module load Python/3.7.0-anaconda-5.3.0-extras-nsc1
$ source activate myEnv
(myEnv) $ ...
```

ENVIRONMENT MANAGEMENT(4): CONDA

Simple examples: cartopy

```
$ module load Python/3.7.0-anaconda-5.3.0-extras-nsc1
$ conda create --name cartopy-eg
$ source activate cartopy-eg
(cartopy-eg) $ conda install cartopy
(cartopy-eg) $ ...
(cartopy-eg) $ source deactivate
$
```

ESMFpy

```
$ module load Python/3.7.0-anaconda-5.3.0-extras-nsc1
$ conda create --name ESMFpy
$ source activate ESMFpy
(ESMFpy) $ conda install -c conda-forge netcdf-fortran
(ESMFpy) $ conda install -c nesii esmpy
(ESMFpy) $ ...
(ESMFpy) $ source deactivate
$
```

pyNGL and pyNIO

```
$ module load Python/3.7.0-anaconda-5.3.0-extras-nsc1
$ conda create --name pyNIO --channel conda-forge pynio pyngl
$ source activate pyNIO
(pyNIO) $ ...
(pyNIO) $ source deactivate
$
```

ENVIRONMENT MANAGEMENT(5): CONDA

Where do the packages end up? By default, packages are installed in

```
~/.conda/pkgs
```

Installing many environments and packages will make this folder grow! But this is a hidden folder.

```
$ du -sh ./.conda/pkgs/
3.9G    ./.conda/pkgs/
$
```

To clean up unused files in this location use

```
$ conda clean -a, --all
$ conda clean -t, --tarballs
$
```

clean --all will remove index cache, lock files, unused cache packages, and tarballs

clean --tarballs will remove cached package tarballs

ENVIRONMENT MANAGEMENT(6): CONDA

To further save space, "archive" environments that you have finished using:

```
$ module load Python/3.7.0-anaconda-5.3.0-extras-nsc1
$ conda create --name myEnv
$ source activate myEnv
(myEnv) $ conda install "SomeProject"
(myEnv) $ conda env export > myEnv.yml
(myEnv) $ source deactivate
$
$ conda remove --name myEnv --all
$
```

```
$ module load Python/3.7.0-anaconda-5.3.0-extras-nsc1
$ conda env create -f myEnv.yml
$ source activate myEnv
...
```

You can also create environments outside /home

```
$ module load Python/3.7.0-anaconda-5.3.0-extras-nsc1
$ conda create -p /proj/bolinc/users/struthers/myCondaEnvs/myEnv
$
```

CONDA

Q. Why does it take so long for conda to install a package?

- Downloading and processing index metadata
- Reducing the index
- Expressing the package data and constraints as a SAT problem
- Running the solver
- Downloading and extracting packages
- Verifying package contents
- Linking packages from package cache into environments

SOME PERSONAL RECOMMENDATIONS

If you are sharing a python script with a colleague/collaborator share the python environment also (see next section)

When to use conda, when to use virtualenv

- pip+virtualenv and conda are (sort of) interchangable, so use the one you are most comfortable with.
- If a *conda install "SomePackage"* is available use conda.
- Remember *pip install "SomePackage"* is available within conda.
- If you want to manage a multi-language software stack (e.g. R, python, perl, ...), use conda.

When not to use conda

- If you need to install a python package that requires compiling, then you should not use a conda environment!
- In this case, you can try to use a virtualenv based on one of the NSC build Python modules and if that fails, then contact support for help (support@nsc.liu.se)

PYTHON: MORE ADVANCED USAGE

LEARNING OBJECTIVES

- Sharing environments
- Jupyter notebooks on Tetralith
- Multi-processor Python jobs

SHARING ENVIRONMENTS: VIRTUALENV (1)

Sharing your environment with a colleague.

```
$ module load Python/3.6.4-nsc2-intel-2018a-eb
$ source myEnv/bin/activate
(myEnv) $ pip freeze > myEnvRequirements.txt
$ deactivate
```

Share your *myEnvRequirements.txt* file with colleagues/collaborators.

```
$ module load Python/3.6.4-nsc2-intel-2018a-eb
$ virtualenv --system-site-packages yourEnv
$ source yourEnv/bin/activate
(yourEnv) $ pip install -r myEnvRequirements.txt
$ deactivate
```

SHARING ENVIRONMENTS: VIRTUALENV (2)

Curating a Python environment within a project.

```
$ module load Python/3.6.4-nsc2-intel-2018a-eb
$ cd /proj/bolinc/shared/software/PythonEnvs/VirtualEnv
$ virtualenv --system-site-packages astropy
$ source astropy/bin/activate
(astropy) $ pip install astropy
(astropy) $ pip install matplotlib
$ deactivate
```

```
$ module load Python/3.6.4-nsc2-intel-2018a-eb
$ source /proj/bolinc/shared/software/PythonEnvs/VirtualEnv/astropy/bin/activate
(astropy) $ ...
$ deactivate
```

- add a README.md in */proj/bolinc/shared/software/PythonEnvs/VirtualEnv/astropy*
- Think about permissions

SHARING ENVIRONMENTS: CONDA (1)

Sharing your environment with a colleague.

```
$ module load Python/3.7.0-anaconda-5.3.0-extras-nsc1
$ conda create --name myEnv
$ source activate myEnv
(myEnv) $ conda env export > myEnvEnvironment.yml
$ source deactivate
```

Share your *myEnvEnvironment.yml* file with colleague/collaborator.

```
$ module load Python/3.7.0-anaconda-5.3.0-extras-nsc1
$ conda env create -f myEnvEnvironment.yml
$ source activate myEnv
(myEnv) $
```

SHARING ENVIRONMENTS: CONDA (2)

Curating a Python environment within a project.

Use *-p PREFIX* instead of *-n NAME* when creating your environment and use the full prefix when activating.

```
$ module load Python/3.7.0-anaconda-5.3.0-extras-nsc1
$ conda create -p /proj/bolinc/shared/software/PythonEnvs/conda/cartopy python=3 cartopy
$
```

assuming you want your environment in */proj/bolinc/shared/software/PythonEnvs/conda/cartopy*.

```
$ source activate /proj/bolinc/shared/software/PythonEnvs/conda/cartopy
(cartopy) $
```

- add a README.md in the */proj/bolinc/shared/software/PythonEnvs/conda/cartopy* folder

JUPYTER NOTEBOOKS (1)

Running a notebook on a login node

Either use Thinlinc

```
$ cd /dir/with/notebook
$ module load Python/3.7.0-anaconda-5.3.0-extras-nsc1
$ jupyter-notebook
```

or if you would rather use a browser on your local laptop/desktop, you can instead use an *ssh tunnel*.

JUPYTER NOTEBOOKS (2)

Running a notebook on a compute node.

- Create an interactive session, load a python module and start a jupyter notebook.

```
$ interactive -N 1 -t 120 -A nsc
salloc: Granted job allocation 5469542
srun: Step created for job 5469542
n348 $
n348 $ module load Python/3.7.0-anaconda-5.3.0-extras-nsc1
n348 $ jupyter-notebook --no-browser --ip=n348
```

- Copy the URL from the interactive terminal on n348 and paste it into your browser.

MULTI-PROCESSOR PYTHON: IPYPARALLEL

Download the examples and start an interactive sesssion.

```
$ git clone https://github.com/DaanVanHauwermeiren/ipyparallel-tutorial.git
$ interactive -n 5 -A nsc -t 01:00:00
```

- create your environment

```
n401 $ module load Python/3.7.0-anaconda-5.3.0-extras-nsc1
n401 $ conda create --name ipyparallel
n401 $ source activate ipyparallel
(ipyparallel) n401 $ conda install ipyparallel
(ipyparallel) n401 $ jupyter serverextension enable --py ipyparallel --user
(ipyparallel) n401 $ jupyter nbextension install --py ipyparallel --user
(ipyparallel) n401 $ jupyter nbextension enable --py ipyparallel --user
(ipyparallel) n401 $ # start your cluster
(ipyparallel) n401 $ ipcluster start --n=4
...
```

- in a second terminal, log into your interactive node

```
$ jobsh n401
n401 $ module load Python/3.7.0-anaconda-5.3.0-extras-nsc1
n401 $ source activate ipyparallel
(ipyparallel) n401 $ jupyter-notebook --no-browser --ip=n401
...
```

- copy and paste the resulting link into a browser

MULTI-PROCESSOR PYTHON

Simple example: mpi4py

- Requires either an interactive session or submission to the batch queue using `sbatch`

```
$ cat mpi4py-batch.sh
#!/bin/sh
#SBATCH -A nsc
#SBATCH -n 4
#SBATCH -t 00:00:30

module load Python/3.6.4-nsc2-intel-2018a-eb
module load buildenv-intel/2018a-eb

mpiexec -n 4 python mpi4py-eg.py
$
$ cat mpi4py-eg.py
from mpi4py import MPI

comm = MPI.COMM_WORLD
rank = comm.Get_rank()

if rank == 0:
    data = {'a': 7, 'b': 3.14}
    comm.send(data, dest=1, tag=11)
elif rank == 1:
    data = comm.recv(source=0, tag=11)
$
$ sbatch mpi4py-batch.sh
Submitted batch job 5524351
$
```

MULTI-PROCESSOR PYTHON

Some more packages that allow for multi-processor

- _thread and threading
- multiprocessing: process based threading interface
- ray
- PP
- pyMPI: integrating the Message Passing Interface (MPI) into the Python interpreter