Python, Pip, Anaconda, and Miniconda HPC:

What is Python?

Python is a high-level programming language with many applications.

To view the current version of python in your PATH:

```
python -V
```

The current interpreter can be located by using:

```
which python
```

We do not recommend using usr/bin/python for building environments.

What is an Environment?

In simple terms, an environment is an isolated instance where one can install different versions and packages of python separate from what is installed in another environment or what is publicly available on the cluster itself. In this case, an environment is important because in an HPC there are many users and most have their own needs and using the base environment (the python interpreter of base system) is not a good choice. For example, a specific piece of legacy code may require a previous version of python that is no longer supported or conflicts with packages installed on the cluster. Therefore, it is important that users can create their own environments in their directory.

What versions of python are available in HPC?

The default version of Python 2.7.5 (usr/bin/python). However, there are multiple versions of python available to work with. You can see them by following command:

```
module avail python
```

As you can see, there are multiple versions of python. You can work with any of them by following:

```
module load PYTHONMODULE
```

where PYTHONMODULE is any path or location displayed when you hit the above module avail command.

For example, if one wants to load Python version 2.5, one can do the following,

```
module load python/python2/2.5
```

This command just loaded the python 2.5 interpreter as the main interpreter. To check this, run the following code:

```
python -V
```

What is Pip?

Pip is a package manager inside python. It helps to install additional packages that are not part of the standard python library. The syntax to install new package using pip is-:

```
pip install <package_name>
```

For example,

```
pip install matplotlib
```

Note: To use pip, it needs to be installed in the python environment.

To install pip in the current environment, use the following command,

```
python -m ensurepip --upgrade
```

What is Conda?

Conda is a package manager, built originally for Python, but can be used to package and distribute software for any major language, e.g., R, Ruby, Lua, Scala, Java, JavaScript, C/ C++ etc. The robustness of Conda makes it appealing to use as an alternative package manager to Pip.

What is Anaconda?

Anaconda is known as an open-source distribution of python and R programming language targeted for data science. The Conda environment comes bundled with Anaconda, a Python interpreter, over 7500 package installations through PyPI or Conda package.

What is Miniconda?

Miniconda is a subset of anaconda which is targeted for scientific computing and engineering. When installed, Conda, Python, and a small number of useful packages including pip, and zlib are installed. This is recommended for people who are using HPC, since it is a minimal installation which will not take up much storage.

How to set up a user python environment using Conda?

Step 1:

The first step is to load the Conda into your session. To do this, you can run this command:

module load miniconda3/base

Alternatively, instead of loading a module, one can source conda by doing the following:

```
source /share/apps/modulefiles/conda_init.sh
```

Now, you should be able to see your conda path with the command: which conda

Step 2:

Next, set up your environment in your home directory.

Quick start:

You can set up the environment with a one liner.

```
conda create -n env_name python=3.9 pip
```

The environment will be in your home directory under a hidden folder called .conda.

To activate an environment

```
conda activate ~/.conda/envs/env_name
```

To test an environment

```
python3 -V          # should have your environment version
which python3       # should be installed in your environment path
```

To deactivate an environment

```
conda deactivate   # this should deactivate your env and send you to base env
```

To delete an environment

```
conda env remove -p PATHTOENV
```

Package Installation:

To install packages on conda environment, you can use either conda package manager or pip package manager.

Note: You must activate your environment first before installing packages to said environment.

The syntax format is:

conda install <package name>

or

pip install <package name>

For example:

conda install numpy

or

pip install numpy

More information here:

https://docs.conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html#creating-an-environment-with-commands

How to load your environment after installation?

```
module load miniconda3/base            # loads conda
```

```
coda env list                            # lists all env

conda activate PATHTOYOURENVIRONMENT     # you can see the path to your env by the
                                          # Second command
```

How to run python file in HPC?

First, load the environment you want to use or use base interpreter. To run a python file on the HPC, you can use these methods.

Contents of test.py:

```
#Calculates the factorial of a positive random integer
import sys
from random import randint, seed
from time import time

def fac(num: int) -> int:
    if num == 0:
        return 1
    else:
        return num * fac(num-1)

seed(time())
x= randint(1, 500)
sys.setrecursionlimit(x**2)
ans = fac(x)
print(f'The factorial of {x} is {ans}')
fl = open('fac.txt', 'w+')
```

```
fl.write(f'The factorial of {x} is {ans}\n--------End of calculation------
--\n')
```

Test with interactive session:

This method puts you on an interactive session on a computer node. This is handy to edit and run the script on the fly, change interpreter version, or install new dependencies while using computer resources. You can load new modules if your scripts depend on other programs.

```
srun -p main --qos main -n 1  --mem-per-cpu=4G --pty  bash          # additional flags can be
#passed
```

This line will redirect you to the compute node. There you can do any tweaking with interpreter or script and execute by using:

```
python3 test.py
```

To exit the compute node to head node, you can exit the session with an exit command.

Test with SBATCH:

This is the preferred way to run a python script. You get additional functionality and can run multiple scripts at once. This method is preferred when your script does not need user-input, which is the case in most of the scientific calculations.

```
#!/bin/bash
#SBATCH -J my_job_name # job name
#SBATCH -n 1       # number of tasks to use (should be 1)
#SBATCH -p main      # partition type
#SBATCH --qos main   # quality of service
#SBATCH --mem-per-cpu=5G     # RAM
#SBATCH -e errors.%A         # error log file
#SBATCH -o output.%A        #STDOUT file
#SBATCH --mail-user={your email address}  # mail the user the status of the job
```

```
# Either load installed python module
module load PYTHONPATH
# Or use your Conda environment
module load miniconda3/base
conda activate path_to_env    #path to env


python3 test.py          # you can use python as well if you loaded python2.xx
```

You can customize a lot with this method like changing the interpreter in the middle of the script or installing packages, changing environment and so on.

Notes about python:

Python has what is called a Global Interpreter Lock which does not allow python to natively use multiprocessing. However, certain libraries like NumPy do computation in C which does have multiprocessing support. So, if you import those libraries, at least computation part done by invoking those libraries, will multi-process. However, if you do not use them, you can learn to use Multiprocessing library which is preinstalled on python to reduce your runtime and explore the potential of our HPC clusters.

More about Multiprocessing