Tutorial: Environment Management

2024 CEDAR Workshop
Snakes on a Spaceship: The Python Maneuver

Leslie Lamarche – Presented by Angeline Burrell

The Fundamentals

- When you install software, it goes somewhere on your computer
- When you ask you computer to run that software, it has to go find it and run it
- Certain default settings or flags define HOW that software is run
- You can have multiple copies of the same software (i.e., python) installed in different places on your computer, each with its own configuration

Environment management saves you from having to keep track of this manually (and type out extremely long file paths).

```
$ /Users/e30737/miniconda3/envs/py310/bin/python myscript.py
```

```
$ python myscript.py
```

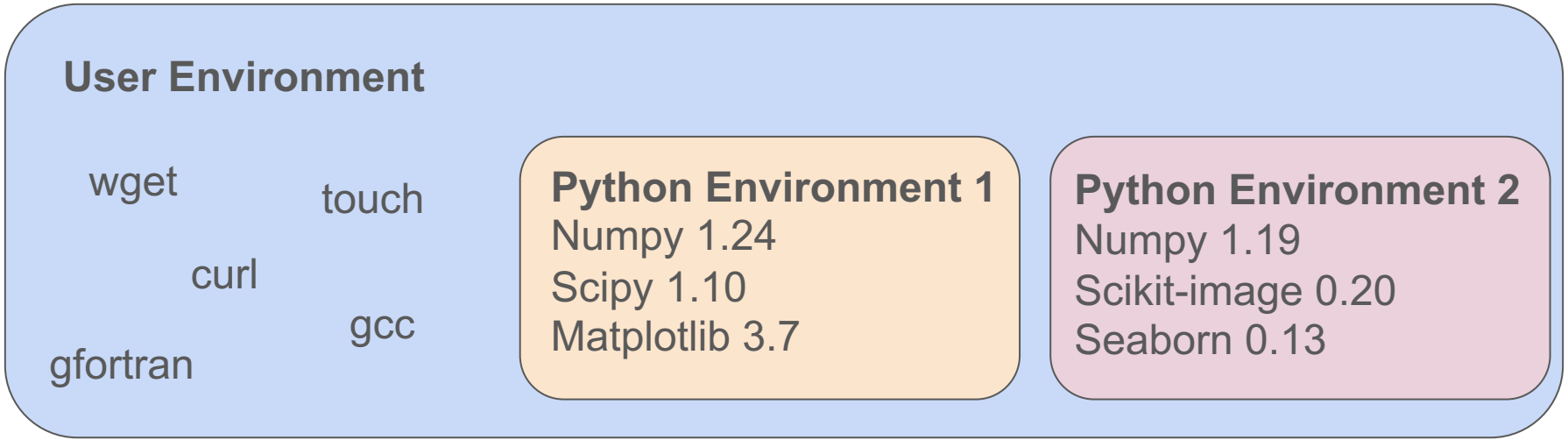
What Are Virtual Environments?

(Definitions from python documentation)

A self-contained directory tree that contains a Python installation for a particular version of Python, plus a number of additional packages.

A cooperatively isolated runtime environment that allows Python users and applications to install and upgrade Python distribution packages without interfering with the behaviour of other Python applications running on the same system.

Environment Conceptualization



When you activate an environment, you are basically preloading a specific set of paths, default functions, and configurations.

Environment Management Software

- Programs that manage environments or the installation and availability of software in these environments
- These can be python-specific, or more general
- IDEs sometimes contain their own environment managers
- Often different package management programs DO NOT play well together, particularly when they cover the same scope

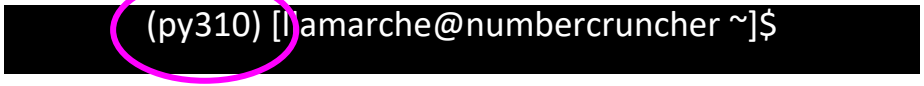
Program	Scope
virtualenvs/pip	Python
conda	Python
apt-get	Ubuntu
chocolatey	Windows
homebrew	Mac
macports	Mac

When are multiple environments useful?

- Create isolated work environments for different things
- Allows you to use different versions of python (and different versions of individual packages)
- Create “standard” environment for all users to run a particular software in to avoid conflicts with other installed packages they might have
- Develop software when you want to maintain a copy of the latest stable release to actually use in your research
- Configure an environment to do something “wierd” without messing up your entire system
- Always have the option to completely delete and environment and rebuild it from scratch if something goes wrong

When can multiple environments be a problem?

- It’s relatively easy to accidentally use the wrong environment and you end up not having access to a package that was installed somewhere else
  - Many terminal applications will put the environment name at the beginning of the terminal prompt, which makes this easy to check



- Not all your software is accessible in one place
  - If you set things up correctly, this is intentional and helpful
- Can create unnecessary duplicates of files on your system (uses extra memory)
  - On most modern systems, the memory volume of source code and executables are trivial
  - Some package management systems handle this more gracefully/intelligently than others

Examples of Different Environments

Modern Workspace
- python 3.12
- numpy
- scipy
- matplotlib

Old Software
- python 2.6
- numpy
- scipy
- basemap

Package Development
- package (development installation)
- package dependencies
- pytest

Prior Workspace
- python 3.9
- numpy
- scipy

Many guides will recommend one environment for each “project”. For research coding, it often is sufficient to have one main environment for your analysis and create others as needed for specific cases.

Common Environment Management Commands

	Virtualenv	Conda
Create New Environment	$ python -m venv <venv>	$ conda create -n <env-name> python=x.x
Activate Environment	$ source <venv>/bin/activate	$ conda activate <env-name>
Install package	$ pip install <package-name>	$ conda install <package-name>
Deactivate Environment	$ deactivate	$ conda deactivate
Delete Environment	$ rm -rf <venv>	$ conda env remove --name <env-name>

<venv> = /path/to/new/virtual/environment/<env-name>

Environment debugging

Fundamentally, most environment management errors come down to you computer not being able to find something or finding the wrong something

- Red flags for environment problems
  - Can't find a program or dependency that you know is installed
  - Persistent errors about incompatible versions
- What to do about it?
  - Make sure you are running your code in the correct environment
  - Use "which" to determine the full path to the executable being run and make sure it's the one you think it is
  - Use "find" to find alternative versions of that program
  - When all else fails, try deleting the entire environment and recreating it from scratch

Many of these problems will still occur (and in fact be WORSE) if you are not using environments

Environment Issues and Sharing Code

- Many of the challenges you encounter trying to share code with other people are related to trying to run code in different environments
  - Code requires a package that's not installed on the local system
  - Code uses python features that are incompatible with the local version
- For "casual" programing, it is not usually practical to completely guard against this, but it's useful to keep it in mind as potential challenges if you're sharing code (or trying to run code that was shared with you)
  - Just because code doesn't work for you immediately doesn't mean it's bad code!
  - Take a look at error messages - the problem may be straight forwards
- For code that you intend for others to use, unit tests are a good way to ensure your code works on multiple systems and in multiple versions of python
  - This a broad topic worthy of its own tutorial

System Python

- Many modern operating systems include a version of python that the OS uses to run internal processes
- NEVER modify this system python
  - Don’t install packages to it
  - Don’t update it
  - In fact, just don’t use it for anything
- It’s there for your SYSTEM to use, NOT you
- Disrupting this may break underlying things your OS is trying to do, which could have unpredictable, confusing, and challenging to reverse impacts
  - MOST things probably won’t cause dire effects, but it’s a possibility so best just to avoid it
- Install python in your own user workspace instead to use for your research

Coordination with Other System Programs

- Sometimes python needs non-python software (i.e., gcc, gfortran, jnode)
- Sometimes these programs are automatically available on your OS, sometimes you'll need to install them manually
- Often your OS will try to be clever and choose what to use because it assumes it knows better than you (i.e., even if you've specifically installed gcc, most MacOS will try to automatically use clang)
- Can be convenient to get you started if you don't really know what you're doing
- Can also cause problems because these programs rely on all kinds underlying libraries that need to be configured carefully and coordinated correctly
- If you choose some things and your OS chooses others and you're not making the same assumptions, things tend to not work
- You can usually use flags to force your computer to behave how you want it to

Jupyter, Kernels, and Virtual Environments

- In order to run jupyter, it must be installed in the environment you are using
- Once running, jupyter uses a kernel to define how python commands are interpreted within that jupyter session
- By default, jupyter automatically creates a dynamic kernel based on the environment it's run out of
  - This ensures there will always be a kernel available for jupyter to use
  - This is fairly intuitive for users
- If you generate and register static kernels in other virtual environments, jupyter can use those instead
  - Even though jupyter is being run out of one virtual environment, you run code and execute commands from a different virtual environment
  - To register a kernel: `$ ipython kernel install --name <kernel-name> --user`

Cloud-based Environments

- A lot of scientific analysis now occurs in cloud-based environments
  - Amazon Web Service
  - Google Colab
  - Other custom or specialized services
- These environments are often initialized with some “sensible” defaults, but are usually configurable and/or customizable to suit your needs
  - May have to spend some time reading detailed documentation
- Who controls the environment?
  - You
  - Whoever set up the particular instance you’re using
  - The cloud service itself
- If all these parties are making different assumptions about what you will and won’t be doing, you’ll probably run into problems
  - Are you only going to be installing python packages, or will you need compilers/other software? Are these available by default, or do you need to add them?
- Cloud-based computing can be a very powerful tool, you just have to use it appropriately for your application

Interactive Development Environments (IDEs)

- Provides a convenient integrated environment where you can edit and run code, view plots, and navigate file structure
  - Spyder
  - Pycharm
  - VCS
  - Atom
- Try to be (too) clever by taking care of a lot of the background business of running python for you
- Most SHOULD have a setting to define which python/python environment you are using
- Some MAY do bits of environment management
  - Install its own version of python
  - Use its own package manager
- What you're managing vs what the IDE is managing gets confusing QUICKLY
  - If you want to use an IDE, ONLY use the IDE
  - The minute you want to do something OUTSIDE the IDE box, you may run into trouble
- When in doubt, close the IDE completely and try running your program in a terminal window
- IDEs are a great way to get started with python/coding, but at some point, you're likely to try to do something that it probably wasn't designed to do very easily
  - Either stop using your IDE
  - Or understand what your IDE is doing well enough that you can be smarter than it is trying to be

Pet Peeve Installation Instructions

Things that probably WILL work, but are dangerous from an environment management perspective and likely to cause confusing problems down the road.

Instruction	What you're actually doing	Why this is problematic
"brew install <required-software>"	Use homebrew to install some software (probably non-python) that is a requirement	You may not typically use homebrew or have it set up, and it may not be compatible with other installation managers you DO use (also this is useless if not on a Mac)
"Create a new virtual environment and install the package"	Install this package in a new, stand-alone virtual environment	You presumably are installing a package to do analysis with your other code, so it's probably not helpful to have it isolated from all your other packages
Directions to modify shell (i.e. bash) profile	Configure your shell session so something (often setting environment variables) is done every time a new shell is started	Sets something up that can override normal environment management which can cause your system to have very confusing default behavior

- Please don't direct users to do these things in your documentation
- If installation instructions tell you to do this, BEWARE (i.e., make sure that's ACTUALLY what you want to do on your system)
- Better Options for Installation
  - Design all python packages so that they can be installed with pip or conda
  - For any non-python dependencies, link to installation instructions provided by that software so users can choose what makes the most sense for their system

Environment Best Practices

- DO create different environments for different purposes, particularly if you work with anything that is fragile
- DO be aware of how your system is constructed in general and where compilers and common shared libraries are pulled from
- DO NOT install things without thinking about how they may impact your system/environments
- AVOID having multiple package managers, particular covering the same scope
- AVOID hacking/working around your chosen environment manager
- DO NOT alter your system python

Main Takeaways

1. Proper environment management is useful for working with a wide variety of software without it conflicting with each other in undesirable ways
2. Understand you may know just enough about how to install things to be dangerous (both to yourself and others)
3. Feel free to use google liberally!
4. You CAN get along for a while not thinking about any of this, but eventually you'll have enough different code on your system that you'll have to deal with it (probably around the second year of your thesis)
5. Be MINDFUL! Think about what you're doing, why you're doing it, and whether or not it makes sense with other things you do.