Python virtual environments

- Introduction - Python@IAC: policy, versions and usage.
- What is a Python virtual environment (VE)?
- What are VEs for: three actual cases.
- Practical example: create and set up a Python VE.

Python @ IAC

- Default Python version in Ubuntu 18.04: v3.6
- Problem: most recent versions of python packages require at least Python 3.7+
- Solution: Installed Python 3.8 and all (general and specialized) modules used by IAC researchers (with dependencies, 650+ in total).
- Choose python version by using modules:
  - No module loaded: default 3.6 version, limited set of packages.
  - module load python/3.6: add all packages installed by us.
  - module load python/3.8: v3.8 (including packages with their most recent version)

What is a Python VE?

- Basically a self-contained Python installation, independent of any other Python virtual environments, and of the system-based Python.
- Solves problems such as incompatible versions among python modules; is immune to updates to system-installed python packages.
- Creation and setup are very straightforward (no need to use Anaconda).
- Can be easily transferred or replicated in a different machine/system (e.g. from Linux to Mac and vv.).
- Ensures portability and reproducibility (your results might change depending on which versions of Python and modules are used).

Problems solved by Python VEs - 1

- Installing “vaex” in the main Python 3.8 environment fails, as it is incompatible with “blake3” (requires an older version).
- Cannot downgrade “blake3” as it would break other packages that depend on it.
- Solution: install “vaex” in a virtual environment, forcing “blake3” to be the older version.
- Activate the environment when needed, deactivate when done.

Problems solved by Python VEs - 2

- Python-based web applications (in our external web server)
- Each was developed with a specific set of packages and versions, and might be incompatible with the system-installed packages.
- Solution: create an independent Python installation (i.e., VE) for each application, and run the application inside its specific VE.

Problems solved by Python VEs - 3

- A user works with the stable version of a python-based pipeline, but at the same time must test the latest development version.
- The two versions cannot coexist in the same Python installation.
- Solution: continue using the stable release installed in the main python installation; create a Python VE for the development version.

Practical example

- Create a Python virtual environment.
- Activate it (and deactivate).
- Install the required packages (fixing version when needed).
- List all installed packages (including dependencies).
- Recreate VE in a different PC, using “freeze” and “requirements”.
- Replicate VE to another, similar machine (same OS and Python version).
- http://venus/SIE/smack_20220705/smack.html
  http://research.iac.es/galeria/ncaon/smack_20220705/smack.html