**Introduction to virtual environments**

Jeffrey M. Perkel, *The sleight-of-hand trick that can simplify scientific computing*, Nature 617, 212-213 (2023).

https://docs.python.org/3/tutorial/venv.html

https://packaging.python.org/tutorials/installing-packages/

https://packaging.python.org/guides/installing-using-pip-and-virtual-environments/

https://virtualenv.pypa.io/en/stable/user\_guide.html

po co – gdy np. potrzebujemy konkretną wersje Pythona / biblioteki – w jednym projekcie jedna wersja, w innym inna

najlepiej zamknąć te wersje w konkretnym venv

**RATIONALE**

Python applications will often use packages and modules that don’t come as part of the standard library. Applications will sometimes need a specific version of a library, because the application may require that a particular bug has been fixed or the application may be written using an obsolete version of the library’s interface.

This means it may not be possible for one Python installation to meet the requirements of every application. The solution for this problem is to create a virtual environment, a self-contained directory tree that contains a Python installation for a particular version of Python, plus a number of additional packages.

Currently, there are two common tools for creating Python virtual environments:

* 'venv' is available by default in Python 3.3 and later, and installs 'pip' and 'setuptools' into created virtual environments in Python 3.4 and later.
* 'virtualenv' needs to be installed separately, but supports Python 2.7+ and Python 3.3+, and 'pip', 'setuptools' and 'wheel' are always installed into created virtual environments by default (regardless of Python version).

It is useful to have a separate directory for virtual environments, e.g. VIRTUAL in the home directory.

**Warning.** Created python virtual environments are usually not self-contained. A complete python packaging is usually made up of thousands of files, so it’s not efficient to install the entire python again into a new folder. Instead virtual environments are mere shells, that contain little within themselves, and borrow most from the system python. This does mean that if you upgrade your system python your virtual environments *might* break.

# Using 'venv' in Linux.

$ python3 -m venv DIR

# DIR can be replaced with ENV\_ML for some machine learning project

$ source DIR/bin/activate

# some actions ...

(DIR) $ deactivate

# Using 'venv' in Windows.

python -m venv DIR

DIR\Scripts\activate

# some actions ...

deactivate

# Using 'virtualenv' in Linux.

$ virtualenv DIR

$ source DIR/bin/activate

# some actions ...

(DIR) $ deactivate

# Using 'virtualenv' in Windows.

virtualenv DIR # the full path may be needed

DIR\Scripts\activate

# some actions ...

deactivate

In order to remove a virtual environment you should simply deactivate it and remove DIR directory (rm -r DIR). You might want to pip freeze a dependency list first.

**PYTHON SCRIPTS IN VIRTUAL ENVIRONMENTS**

#!/usr/bin/env python # or 'python3' if you have Py2 and Py3

#

# hello.py

print("Hello!")

(DIR) $ python hello.py # it is safe for Py2 and Py3

Hello!

(DIR) $ chmod a+x hello.py

(DIR) $ ./hello.py # problems with '#!/usr/bin/python'

Hello!

(DIR) $

**Using pip**

https://pip.pypa.io/en/stable/   
pip documentation

https://packaging.python.org/tutorials/installing-packages/

**INTRODUCTION**

'pip' is the package installer for Python. You can use it to install packages from the Python Package Index (PyPI) and other indexes.

In Debian 10 Linux, there is 'pip' (Py2) and 'pip3' (Py3).

'pip' is already installed if you are using Python 2 >= 2.7.9 or Python 3 >= 3.4 downloaded from python.org or if you are working in a virtual environment created by 'virtualenv' or 'venv'.

# Testing if 'pip' is installed.

$ pip --version # Debian 10

pip 18.1 from /usr/lib/python2.7/dist-packages/pip (python 2.7)

$ pip3 --version # Debian 10

pip 18.1 from /usr/lib/python3/dist-packages/pip (python 3.7)

$ python -m pip --version # Py2, from pip documentation

$ python3 -m pip --version # Py3, note the name 'pip', not 'pip3' here

$ man pip # help in Debian Linux

$ pip --help # usage and commands

**INSTALLING PIP**

APT (for the entire computer)

Debian packages: python-pip, python-wheel, python-pip-whl,

python3-pip, python3-wheel, and dependencies.

**QUICKSTART**

$ python3 -m pip install somepackage # latest version

$ python3 -m pip install somepackage==1.0.4 # specific version

$ python3 -m pip install "somepackage==1.0.4" # specific version

$ python3 -m pip install "somepackage>=1,<2"

$ python3 -m pip install --upgrade somepackage # -U, --upgrade

$ python3 -m pip install somepackage-1.0-py3-none-any.whl # from a wheel archive

$ python3 -m pip install somepackage-1.0.tar.gz # from a tar.gz archive

$ python3 -m pip show somepackage # info about a package

$ python3 -m pip show --files somepackage # show what files were installed

$ python3 -m pip list # the list of packages installed

$ python3 -m pip list --outdated # errors in Debian

$ python3 -m pip uninstall somepackage

$ python3 -m pip check # checking a system

# No broken requirements found.

$ python3 -m pip search "query" # errors in Debian

# Upgrading pip - it is safe in virtual environments.

# For installig packages from binary archives

$ python3 -m pip install --upgrade pip

# For installing packages from source archives

$ python3 -m pip install --upgrade pip setuptools wheel

**REPRODUCIBLE INSTALLS**

As libraries get updated, results from running your code can change, or your code can break completely. It’s important to be able to reconstruct the set of packages and versions you’re using. Best practice is to:   
(1) use a different environment per project you’re working on,   
(2) record package names and versions using pip in 'requirements files'.

# This is done typically in virtual environments.

$ python3 -m pip freeze > requirements.txt # output installed packages in requirements format

$ python3 -m pip install -r requirements.txt # -r, --requirement

**Using venv**

https://docs.python.org/3/library/venv.html   
venv - Creation of virtual environments

**INTRODUCTION**

The 'venv' module provides support for creating lightweight virtual environments with their own site directories. Each virtual environment has its own Python binary (which matches the version of the binary that was used to create this environment) and can have its own independent set of installed Python packages in its site directories.

# Debian 10 apt package - python3-venv

$ python3 -m venv VE1 # creating the virtual environment in the VE1 directory

$ ls VE1

bin include lib lib64 pyvenv.cfg share

$ cat VE1/pyvenv.cfg

home = /usr/bin

include-system-site-packages = false

version = 3.7.3

# Windows 10

c:\> python -m venv c:\path\to\myenv

# Suggestion: use the directory c:\VIRTUAL\VE1

This will create the 'VE1' directory if it doesn’t exist, and also create directories inside it containing a copy of the Python interpreter, the standard library, and various supporting files. A common directory location for a virtual environment is '.venv'. Once you’ve created a virtual environment, you may activate it.

$ source VE1/bin/activate # for the bash shell in Linux

# source VE1/bin/activate.csh # for the tcsh shell in Linux

# C:\> EV1\Scripts\activate.bat # for cmd.exe in Windows

# PS C:\> EV1\Scripts\Activate.ps1 # for PowerShell in Windows

(VE1) $ python

Python 3.7.3 (default, Jan 22 2021, 20:04:44)

[GCC 8.3.0] on linux

Type "help", "copyright", "credits" or "license" for more information.

>>> import sys

>>> sys.executable

'/home/andrzej/VIRTUAL/VE1/bin/python'

>>> quit()

(VE1) $ pip list

Package Version

------------- -------

pip 18.1

pkg-resources 0.0.0

setuptools 40.8.0

(VE1) $ pip install PrettyTable

Collecting PrettyTable

Downloading https://files.pythonhosted.org/packages/26/1b/42b59a4038bc0442e3a0085bc0de385658131eef8a88946333f870559b09/prettytable-2.1.0-py3-none-any.whl

Collecting wcwidth (from PrettyTable)

Downloading https://files.pythonhosted.org/packages/59/7c/e39aca596badaf1b78e8f547c807b04dae603a433d3e7a7e04d67f2ef3e5/wcwidth-0.2.5-py2.py3-none-any.whl

Collecting importlib-metadata; python\_version < "3.8" (from PrettyTable)

Downloading https://files.pythonhosted.org/packages/8e/e2/49966924c93909d47612bb47d911448140a2f6c1390aec2f4c1afbe3748f/importlib\_metadata-4.0.1-py3-none-any.whl

Collecting zipp>=0.5 (from importlib-metadata; python\_version < "3.8"->PrettyTable)

Downloading https://files.pythonhosted.org/packages/0f/8c/715c54e9e34c0c4820f616a913a7de3337d0cd79074dd1bed4dd840f16ae/zipp-3.4.1-py3-none-any.whl

Collecting typing-extensions>=3.6.4; python\_version < "3.8" (from importlib-metadata; python\_version < "3.8"->PrettyTable)

Downloading https://files.pythonhosted.org/packages/2e/35/6c4fff5ab443b57116cb1aad46421fb719bed2825664e8fe77d66d99bcbc/typing\_extensions-3.10.0.0-py3-none-any.whl

Installing collected packages: wcwidth, zipp, typing-extensions, importlib-metadata, PrettyTable

Successfully installed PrettyTable-2.1.0 importlib-metadata-4.0.1 typing-extensions-3.10.0.0 wcwidth-0.2.5 zipp-3.4.1

(VE1) $ pip list

Package Version

------------------ --------

importlib-metadata 4.0.1

pip 18.1

pkg-resources 0.0.0

prettytable 2.1.0

setuptools 40.8.0

typing-extensions 3.10.0.0

wcwidth 0.2.5

zipp 3.4.1

(VE1) $ deactivate

$

**Using virtualenv**

https://virtualenv.pypa.io/en/stable/

**INTRODUCTION**

'virtualenv' is a tool to create isolated Python environments. Since Python 3.3, a subset of it has been integrated into the standard library under the 'venv' module. The 'venv' module does not offer all features of this library:   
(a) is slower,   
(b) is not as extendable,   
(c) cannot create virtual environments for arbitrarily installed python versions (and automatically discover these),   
(d) is not upgrade-able via 'pip',   
(e) does not have as rich programmatic API.

**INSTALLING VIRTUALENV**

pip install virtualenv

In Debian Linux we have python-virtualenv, python3-virtualenv, virtualenv.

**USING VIRTUALENV**

$ virtualenv --version # testing version in Debian 10

15.1.0

$ virtualenv ENV1 # python2 will be used

Running virtualenv with interpreter /usr/bin/python2

New python executable in /home/andrzej/VIRTUAL/ENV1/bin/python2

Also creating executable in /home/andrzej/VIRTUAL/ENV1/bin/python

Installing setuptools, pkg\_resources, pip, wheel...done.

$ source ENV1/bin/activate

# source ENV1/bin/activate.csh # for the tcsh shell in Linux

# C:\> ENV1\Scripts\activate.bat # for cmd.exe in Windows

# PS C:\> ENV1\Scripts\Activate.ps1 # for PowerShell in Windows

(ENV1) $ python --version

Python 2.7.16

(ENV1) $ which python

/home/andrzej/VIRTUAL/ENV1/bin/python

(ENV1) $ which pip

/home/andrzej/VIRTUAL/ENV1/bin/pip

(ENV1) $ pip list

DEPRECATION: Python 2.7 reached the end of its life ...

Package Version

------------- -------

pip 20.3.4 # different from Debian pip!

pkg-resources 0.0.0

setuptools 44.1.1

wheel 0.36.2

(ENV1) $ deactivate

$ virtualenv -p python3 ENV2 # python3 will be used

Already using interpreter /usr/bin/python3

Using base prefix '/usr'

New python executable in /home/andrzej/VIRTUAL/ENV2/bin/python3

Also creating executable in /home/andrzej/VIRTUAL/ENV2/bin/python

Installing setuptools, pkg\_resources, pip, wheel...done.

$ source ENV2/bin/activate

(ENV2) $ which python

/home/andrzej/VIRTUAL/ENV2/bin/python

(ENV2) $ python --version

Python 3.7.3

(ENV2) $ pip --version

pip 21.1 from /home/andrzej/VIRTUAL/ENV2/lib/python3.7/site-packages/pip (python 3.7)

(ENV2) $ pip list

Package Version

------------- -------

pip 21.1 # latest, different from Debian pip3!

pkg-resources 0.0.0

setuptools 56.0.0

wheel 0.36.2

(ENV2) $ deactivate