PYTHON VIRTUAL ENVIRONMENTS

For today's short session, we will perform steps to create a virtual Python environment.

You may opt to *merely observe* or *play along*.

To play along, you will need a Python 3.6 or later. That's it!

Sit back, relax, and we'll get started shortly.

--Rob

PYTHON VIRTUAL ENVIRONMENTS

- Python virtual environment and package management tools are numerous *and perhaps a little confusing*
  - *virtualenv*
  - *pyvenv*
  - *venv*
  - *pyenv*
  - *Pipenv*
  - *Others*

- A virtual environment *allows for managing different versions of packages* without conflicting with the primary Python installation

TOOLS TO MANAGE VIRTUAL ENVIRONMENTS

- Python provides the ability to create separate installations (called virtual environments) to manage third-party packages
  - virtualenv was one the first popular tools to be used for both Python 2 and 3 *(even today!)*
    - It needs to be *pip* installed to use
  - pyvenv was used in early Python 3 versions
    - It had several issues and never stepped out of virtualenv's shadow and is not used now
  - venv became the official virtualization tool starting in Python 3.6
    - It ships with all current versions of Python and is the primary utility used

VENV

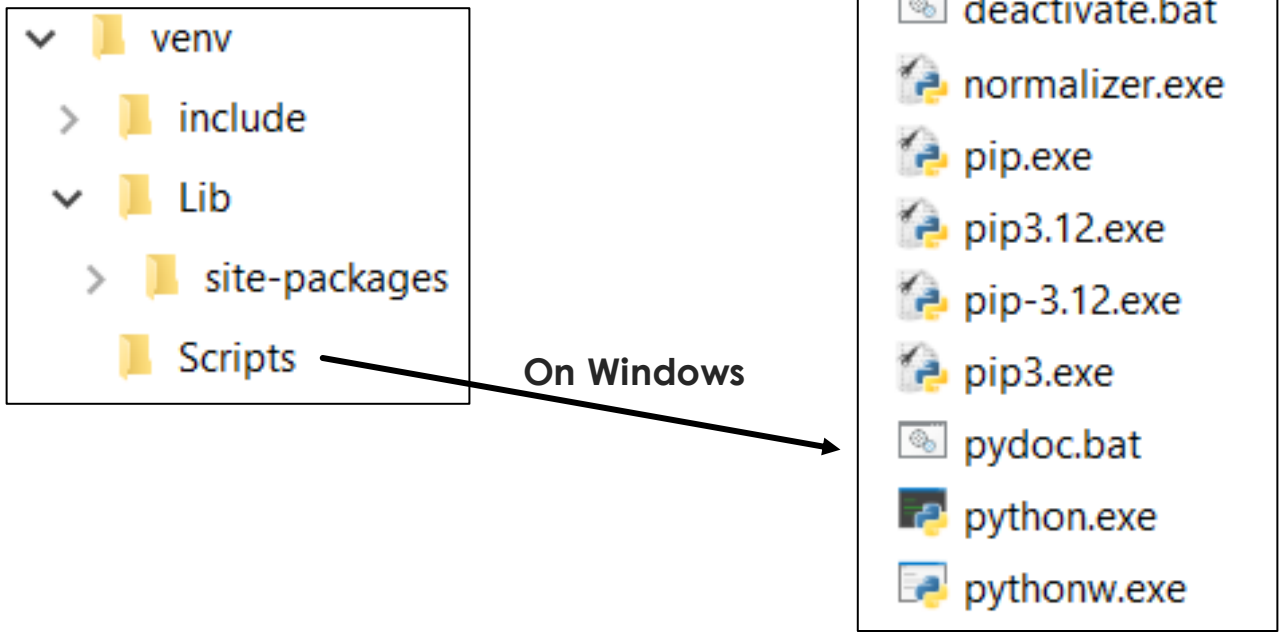
The new directory gets created at the location where the command runs

Specify python, python3, python3.12, etc., (as needed) when running venv

```
python -m venv relative_or_absolute_path
```

- This command creates a new virtual environment at the specified location

- Once created, the environment needs to become the primary one by "*activating*" it



ACTIVATION / DEACTIVATION

- To enable the virtual environment, run the *activate* command
  - On Windows: venv\Scripts\activate
  - On OS X: source ./venv/bin/activate
- When finished, issue the *deactivate* command to end use of the virtual environment or simply close the terminal window
- To *remove* the virtual environment, first deactivate, then *delete the venv* directory
  - Optionally, use rm -rf venv on OS X

What actually happens when you activate?

FREEZE AND REQUIREMENTS.TXT

```
pip freeze > requirements.txt
```

- Use the *freeze* command to identify the contents of currently installed packages

```
Flask==3.0.0
Jinja2==3.1.2
SQLAlchemy==2.0.22
beautifulsoup4==4.12.2
colorama==0.4.6
prettytable==3.9.0
requests==2.31.0
wcwidth==0.2.9
```

requirements.txt can be generated based on the currently installed set of packages

- *pip* can be used to install packages from a *requirements.txt* file

```
pip install -r requirements.txt
```

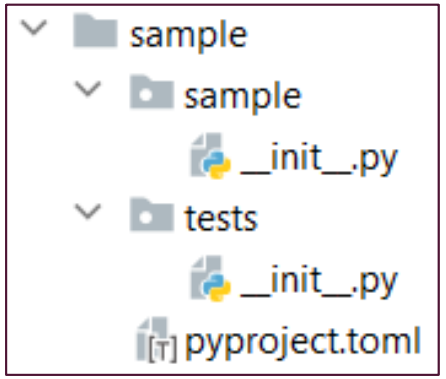
POETRY

```
pip install poetry
```

- Poetry is a multi-purpose project manager for Python
  - It supports multiple subcommands for performing different actions

```
poetry new sample
```

Creates a new project with this structure



Symbols allowed:

"1.2.3"	exact version only
"^1.2.3"	Version up to leftmost non-zero value (1.2.3 >= x < 2.0.0)
"~1.2.3"	Changes up through least significant digit (1.2.3 >= x < 1.3.0)
">=1.2.3"	Can also use >, <, <=, !=
"1.2.*"	Latest version in the position indicated (1.2.0 >= x < 1.3.0)

```
[tool.poetry]
name = "sample"
version = "0.1.0"
description = ""
authors = ["Your Name <you@example.com>"]
readme = "README.md"

[tool.poetry.dependencies]
python = "^3.12"
requests = "*"
prettytable = "*'

[tool.poetry.dev-dependencies]
pytest = "^7.4.3"

[build-system]
requires = ["poetry-core"]
build-backend = "poetry.core.masonry.api"
```

Identify basic project info

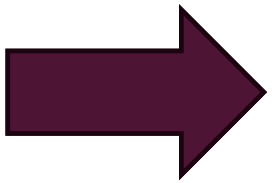
Add packages into the .toml file

POETRY *(CONTINUED)*

poetry env use *path/to/python_exe*

poetry env use creates a virtual python environment in a pre-defined directory depending on the OS:
Windows: C:\Users\<username>\AppData\Local\pypoetry\Cache
OS X: ~/Library/Caches/pypoetry
Linux: ~/.cache/pypoetry

- To create a virtual environment, issue from within the project directory poetry install
- To install the dependencies defined in the *.toml* file, run



Creates a *poetry.lock* file containing exact versions installed

poetry.lock

```
...

[[package]]
name = "prettytable"
version = "3.9.0"
...

[[package]]
name = "requests"
version = "2.31.0"
```

poetry shell - activates the new environment
exit - exits the shell
poetry env remove <name of poetry_env>
- removes the virtual environment

OTHER TOOLS

- pyenv is a Python *version* switching tool for Unix/Linux

```
pip install pyenv
```

  - A Windows version exists as well

```
pip install pyenv-win
```

```
$ pyenv install 3.12     # installs latest 3.12 version
$ pyenv versions         # list installed versions
$ pyenv global 3.10.5    # selects local version
$ pyenv local 3.11.3     # selects global version
```

- pipenv is another package management tool designed to emulate Ruby and Node.js environments

```
pip install pipenv
```

  - Use pipenv install to create an environment at the location where you run the command
    - Run pipenv shell to activate
    - Deactivate (afterwards) by exiting the shell