Why would we use a virtual environment?

The purpose of a virtual environment is to have a space where we can install packages that are specific to a certain project.

**PIPENV**

* pipfile:
* pipfile.lock: includes the hashes of packages. It contains the version of packages and their dependencies. This is a **deterministic** list of your virtual environment, so everytime your env can be the same. You can decide to update versions after you are sure the codes work.

**pipenv install** install the virtual environment based on the pipfile

**pipenv install <library>** install a third party library in the virtual environment

**pipenv install -r <**C:\dev\Exposures\requirements.txt**>** install a requirement file

**pipenv uninstall <library>** uninstall a third party library in the virtual environment

**pipenv install <library> --dev** install a package in a dev environment (but not showing audience that it exists)

**pipenv shell** activate the virtual environment

**pipenv run python** run a python file using pipenv but without activating the pipenv

**exit()** to deactivate the virtual environment

**pipenv lock** update the pipenv.lock

**pipenv lock -r** display current installed packages and dependencies in a requirements.txt

**pipenv --python 3.6** If you want to use a different version of python in pipenv, go to pipfile and change the version to what you want. Run pipenv --python 3.6

**pipenv --rm** remove the virtual environment completely.

**pipenv --venv** look the virtual env location

**pipenv check** check installed package safety. You can modify the version in pipfile and run pipenv install again

**pipenv graph** check what package depends on what

**pipenv install --ignore-pipfile** create the environment using the list in pipfile.lock and ignore the pipfile that is usually used by default

‘venv’ is a standard module in Python that works with virtual environment (python 3.3 or above)

In the command prompt:

code . open visual studio code

**H:\>** C: & **H:\>** cd Users\yu-shenglee\Desktop: change current directory to Desktop

**pip list** to show the current installed packages in the system(globally)

**C:\Users\yu-shenglee\Desktop>** python -m venv project\_env (create a virtual environment where project\_env is the environment name)

**C:\Users\yu-shenglee\Desktop>** dir (show objects on the directory, you should see the project\_env)

**C:\Users\yu-shenglee\Desktop>** project\_env\Scripts\activate.bat Activate the environment file

**(project\_env) C:\Users\yu-shenglee\Desktop>** where python Give the path of the current python command

**(project\_env) C:\Users\yu-shenglee\Desktop>** pip install requests To install packages in this particular environment

**(project\_env) C:\Users\yu-shenglee\Desktop>** pip list To see installed packages in this particular environment

**(project\_env) C:\Users\yu-shenglee\Desktop>** pip freeze To see installed packages in this particular environment, put info in a requirements.txt file

If you want to export the packages you are using for a certain environment. The way we can do this in python is with a requirement.txt file. This would allow someone else to create an environment and use your requirement.txt file to install all of the same packages and dependencies that you are using.

**(project\_env) C:\Users\yu-shenglee\Desktop>** deactivate Deactivate an environment

**C:\Users\yu-shenglee\Desktop>** rmdir project\_env /s To delete the entire tree of project\_env

**C:\Users\yu-shenglee\Desktop>** mkdir my\_project To make a directory ‘my\_project’ on desktop

**C:\Users\yu-shenglee\Desktop>** python -m venv my\_project\venv To create a venv file inside the my\_project folder on the desktop

**C:\Users\yu-shenglee\Desktop>** my\_project\venv\Scripts\activate.bat To activate the created virtual environment

**(venv) C:\Users\yu-shenglee\Desktop>** pip install -r requirements.txt To install everything from that requirement.txt file to venv

**(venv) C:\Users\yu-shenglee\Desktop>** pip list Should show everything installed including the requirements.txt

Don’t save files inside that venv folder !!

**Creating an environment with access to the system packages:**

**C:\Users\yu-shenglee\Desktop\Python>** python -m venv venv364 --system-site-packages

**C:\Users\yu-shenglee\Desktop\Python>** venv364\Scripts\activate.bat

**(venv 364) C:\Users\yu-shenglee\Desktop\Python>** pip list (The package that we install in this environment won’t affect system packages)

**(venv 364) C:\Users\yu-shenglee\Desktop\Python>** pip install SQLalchemy (will only be in venv364)

**(venv 364) C:\Users\yu-shenglee\Desktop\Python>** pip list –local (show packages that are only installed in venv364, but not system!!)

**(venv 364) C:\Users\yu-shenglee\Desktop\Python>** deactivate

**(venv 364) C:\Users\yu-shenglee\Desktop\Python>** pip freeze –local

C:\dev\Production>pipenv -h

Usage: pipenv [OPTIONS] COMMAND [ARGS]...

Options:

--where Output project home information.

--venv Output virtualenv information.

--py Output Python interpreter information.

--envs Output Environment Variable options.

--rm Remove the virtualenv.

--bare Minimal output.

--man Display manpage.

--support Output diagnostic information for use in

GitHub issues.

--site-packages / --no-site-packages

Enable site-packages for the virtualenv.

[env var: PIPENV\_SITE\_PACKAGES]

--python TEXT Specify which version of Python virtualenv

should use.

--three / --two Use Python 3/2 when creating virtualenv.

--clear Clears caches (pipenv, pip). [env var:

PIPENV\_CLEAR]

-v, --verbose Verbose mode.

--pypi-mirror TEXT Specify a PyPI mirror.

--version Show the version and exit.

-h, --help Show this message and exit.

Usage Examples:

Create a new project using Python 3.7, specifically:

$ pipenv --python 3.7

Remove project virtualenv (inferred from current directory):

$ pipenv --rm

Install all dependencies for a project (including dev):

$ pipenv install --dev

Create a lockfile containing pre-releases:

$ pipenv lock --pre

Show a graph of your installed dependencies:

$ pipenv graph

Check your installed dependencies for security vulnerabilities:

$ pipenv check

Install a local setup.py into your virtual environment/Pipfile:

$ pipenv install -e .

Use a lower-level pip command:

$ pipenv run pip freeze

Commands:

check Checks for PyUp Safety security vulnerabilities and against PEP

508 markers provided in Pipfile.

clean Uninstalls all packages not specified in Pipfile.lock.

graph Displays currently-installed dependency graph information.

install Installs provided packages and adds them to Pipfile, or (if no

packages are given), installs all packages from Pipfile.

lock Generates Pipfile.lock.

open View a given module in your editor.

run Spawns a command installed into the virtualenv.

scripts Lists scripts in current environment config.

shell Spawns a shell within the virtualenv.

sync Installs all packages specified in Pipfile.lock.

uninstall Uninstalls a provided package and removes it from Pipfile.

update Runs lock, then sync.