Packages and Environments

Packages

There are two meanings for "package" in Python:

1. Subdirectories into which modules are organized. See Python's module documentation for details.
2. A distribution of 3rd-party software, e.g., Python modules and supporting files, native code, etc.

Here we discuss the second meaning.

Installing Packages

The `pip3` command downloads and installs packages.

- Packages come from the Python Package Index by default.
- `pip3` is quite flexible, allowing you to install from many kinds of sources. See the Python package tutorial for details.

You can invoke `pip3` in two ways, for example, to install `ipython`:

```
python3 -m pip install ipython
```

or

```
pip3 install ipython
```

Module Search Path

Just as an operating system command shell searches for executable programs by searching the directories listed in the `PATH` environment variable, Python finds modules by searching directories. The module search path is stored in `sys.path`:

```
>>> import sys
>>> from pprint import pprint
>>> pprint(sys.path)
['',
 '/Library/Frameworks/Python.framework/Versions/3.10/lib/python310.zip',
 '/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10',
 '/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/lib-dynload',
 '/Users/drcs/vcs/github.com/drcscodes/python-coursework/venv/lib/python3.10/site-pa
```

- Notice that the current directory, represented by the `''` at the beginning of the search path, is part of `sys.path`, which is why you can import modules located in your current directory.
- Notice that our virtual environment is in the `sys.path`.
- Note use of `pprint`, which "pretty prints" the `sys.path` list in a more easily readable format.

Virtual Environments

Different Python projects may use different versions of the same package. To avoid conflicts, use virtual environments.

In the root directory of your Python project, create your virtual environment with:

```
python3 -m venv venv
```

This creates a virtual environment in the `venv` subdirectory of your project root directory. Activate the virtual environment on macOS or Linux with:

```
source venv/bin/activate
```

or in Windows PowerShell (if this doesn't work, try `Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser`. See venv docs for details.):

```
venv\bin\activate.ps1
```

Deactivate a virtual environment with (macOS, Linux, or Windows):

```
deactivate
```

Active Review

- Create a scratch directory, `cd` to it, and then create a virtual environment in the `test` subdirectory.
- Print the contents of `sys.path`.
  - Tip: you can do that quickly with `python3 -c "import sys, pprint as pp; pp.pprint(sys.path)"`
- Activate your new `test` environment.
  - Note the difference in your OS command shell's prompt string, if any.
- Print the contents of `sys.path` again and note the difference.
- Deactivate your `test` environment.
  - You can delete the scratch directory.
- In your OS command shell, repeat the steps above, except for the first step of creating an environment, for the `venv` directory created by a PyCharm project.
- In PyCharm with a Python project open, open a terminal and note the automatic loading of its `venv` environment.

requirements.txt

In your Python projects you should include a requirements.txt file in the root directory of your project and add requirements.txt to your project's Git repository. With your virtual environment activated and all required packages installed, create requirements.txt with:

```
python3 -m pip freeze > requirements.txt
```

Be sure to re-run that command and update in Git whenever you add new dependencies. When another programmer clones your project's repository, they can create a virtual environment and install all the required dependencies into it with:

```
python3 -m pip install -r requirements.txt
```

Take a look at a few prominent OSS Python projects and notice that they all have a requirements.txt in the project root directory.

- https://github.com/ansible/ansible
- https://github.com/numpy/numpy – multiple task-specific requirements files
- https://github.com/pandas-dev/pandas
- https://github.com/keras-team/keras
- https://github.com/pytorch/pytorch

Conclusion

- Use virtual environments to manage dependencies in Python projects.
- Each project should have its own virtual environment, stored in a subdirectory of the project root directory.
  - The overhead of a few megabytes to a few tens of megabytes is repaid many times over in reduced complexity.
- Don't forget to activate and deactivate environments.
- Document your project's dependencies with a `requirements.txt` file.