Using Python libraries

This lab covers

- [Developing and debugging tools]
- [Accessing PyPI (a.k.a. "The Cheese Shop")]
- [Installing Python libraries and virtual environments using pip and venv]

Installing Python libraries using pip and venv

If you need a third-party module that isn't prepackaged for your platform, you'll have to turn to its source distribution. This fact presents a couple of problems:

- [To install the module, you must find and download it.]
- [Installing even a single Python module correctly can involve a certain amount of hassle in dealing with Python's paths and your system's permissions, which makes a standard installation system helpful.]

Python offers [pip] as the current solution to both problems. [pip] tries to find the module in the Python Package index (more about that soon), downloads it and any dependencies, and takes care of the installation. The basic syntax of [pip] is quite simple. To install the popular requests library from the command line, for example, all you have to do is

```
$ python3 -m pip install requests
```

Upgrading to the library's latest version requires only the addition of the [---upgrade] switch:

```
$ python3 -m pip install --upgrade requests
```

Finally, if you need to specify a particular version of a package, you can append it to the name like this:

```
$ python3 -m pip install requests==2.11.1
$ python3 -m pip install requests>=2.9
```

Installing with the --user flag

On many occasions, you can't or don't want to install a Python package in the main system instance of Python. Maybe you need a bleeding-edge version of the library, but some other application (or the system itself) still uses an older version. Or maybe you don't have access privileges to modify the system's default Python. In cases like those, one answer is to install the library with the [---user] flag. This flag installs the library in the user's home directory, where it's not accessible by any other users. To install [requests] for only the local user:

```
$ python3 -m pip install --user requests
```

As I mentioned previously, this scheme is particularly useful if you're working on a system on which you don't have sufficient administrator rights to install software, or if you want to install a different version of a module. If your needs go beyond the basic installation methods discussed here, a good place to start is "Installing Python Modules," which you can find in the Python documentation.

Virtual environments

You have another, better option if you need to avoid installing libraries in the system Python. This option is called a virtual environment (virtualenv). A *virtual environment* is a self-contained directory structure that contains both an installation of Python and its additional packages. Because the entire Python environment is contained in the virtual environment, the libraries and modules installed there can't conflict with those in the main system or in other virtual environments, allowing different applications to use different versions on both Python and its packages.

Creating and using a virtual environment takes two steps. First, you create the environment:

```
$ python3 -m venv test-env
```

This step creates the environment with Python and [pip] installed in a directory called test-env. Then, when the environment is created, you activate it. On Windows, you do this:

```
> test-env\Scripts\activate.bat
```

On Unix or MacOS systems, you source the activate script:

```
$ source test-env/bin/activate
```

When you've activated the environment, you can use [pip] to manage packages as earlier, but in the virtual environment [pip] is a standalone command:

```
$ pip install requests
```

In addition, whatever version of Python you used to create the environment is the default Python for that environment, so you can use just [python] instead of [python3] or [python3.6].

Virtual environments are very useful for managing projects and their dependencies and are very much a standard practice, particularly for developers working on multiple projects. For more information, look at the "[Virtual Environments and Packages]" section of the Python tutorial in the Python online documentation.

Summary

- [Python has a rich standard library that covers more common situations than many other languages, and you should check what's in the standard library carefully before looking for external modules.]
- [If you do need an external module, prebuilt packages for your operating system are the easiest option, but they're sometimes older and often hard to find.]
- [The standard way to install from source is to use pip, and the best way to prevent conflicts among multiple projects is to create virtual environments with the venv module.]
- [Usually, the logical first step in searching for external modules is the Python Package Index (PyPI).]