# About virtualenv

As you probably have guessed by its name, **virtualenv** is all about virtual environments. Let me explain what they are and why we need them and let me do it by means of a simple example.

You install Python on your system and you start working on a website for client X. You create a project folder and start coding. Along the way you also install some libraries, for example the Django framework, which we'll see in depth in *Chapter 10*, *Web Development Done Right*. Let's say the Django version you install for project X is 1.7.1.

Now, your website is so good that you get another client, Y. He wants you to build another website, so you start project Y and, along the way, you need to install Django again. The only issue is that now the Django version is 1.8 and you cannot install it on your system because this would replace the version you installed for project X. You don't want to risk introducing incompatibility issues, so you have two choices: either you stick with the version you have currently on your machine, or you upgrade it and make sure the first project is still fully working correctly with the new version.

Let's be honest, neither of these options is very appealing, right? Definitely not. So, here's the solution: virtualenv!

virtualenv is a tool that allows you to create a virtual environment. In other words, it is a tool to create isolated Python environments, each of which is a folder that contains all the necessary executables to use the packages that a Python project would need (think of packages as libraries for the time being).

So you create a virtual environment for project X, install all the dependencies, and then you create a virtual environment for project Y, installing all its dependencies without the slightest worry because every library you install ends up within the boundaries of the appropriate virtual environment. In our example, project X will hold Django 1.7.1, while project Y will hold Django 1.8.

It is of vital importance that you never install libraries directly at the system level. Linux for example relies on Python for many different tasks and operations, and if you fiddle with the system installation of Python, you risk compromising the integrity of the whole system (guess to whom this happened…). So take this as a rule, such as brushing your teeth before going to bed: always, always create a virtual environment when you start a new project.

To install virtualenv on your system, there are a few different ways. On a Debian-based distribution of Linux for example, you can install it with the following command:

**$ sudo apt-get install python-virtualenv**

Probably, the easiest way is to use pip though, with the following command:

**$ sudo pip install virtualenv # sudo may by optional**

pip is a package management system used to install and manage software packages written in Python.

Python 3 has built-in support for virtual environments, but in practice, the external libraries are still the default on production systems. If you have trouble getting virtualenv up and running, please refer to the virtualenv official website: <https://virtualenv.pypa.io>.

Source: “Learning Python”, Fabrizio Romano, PACKT Publishing.

# Install virtual environment on Windows 7

There are five steps involved in installing the virtual environment on Windows.

1. Create folder
2. Create virtual environment
3. Activate virtual environment
4. Verify which Python
5. Deactivate virtual environment.
6. >cd D:\users\uriplo\

>mkdir venv

1. #Program Files\Python36\Scripts\>virtualenv D:\users\uriplo\venv
2. #D:\users\uriplo\venv\Scripts\>activate
3. >where
4. >deactivate