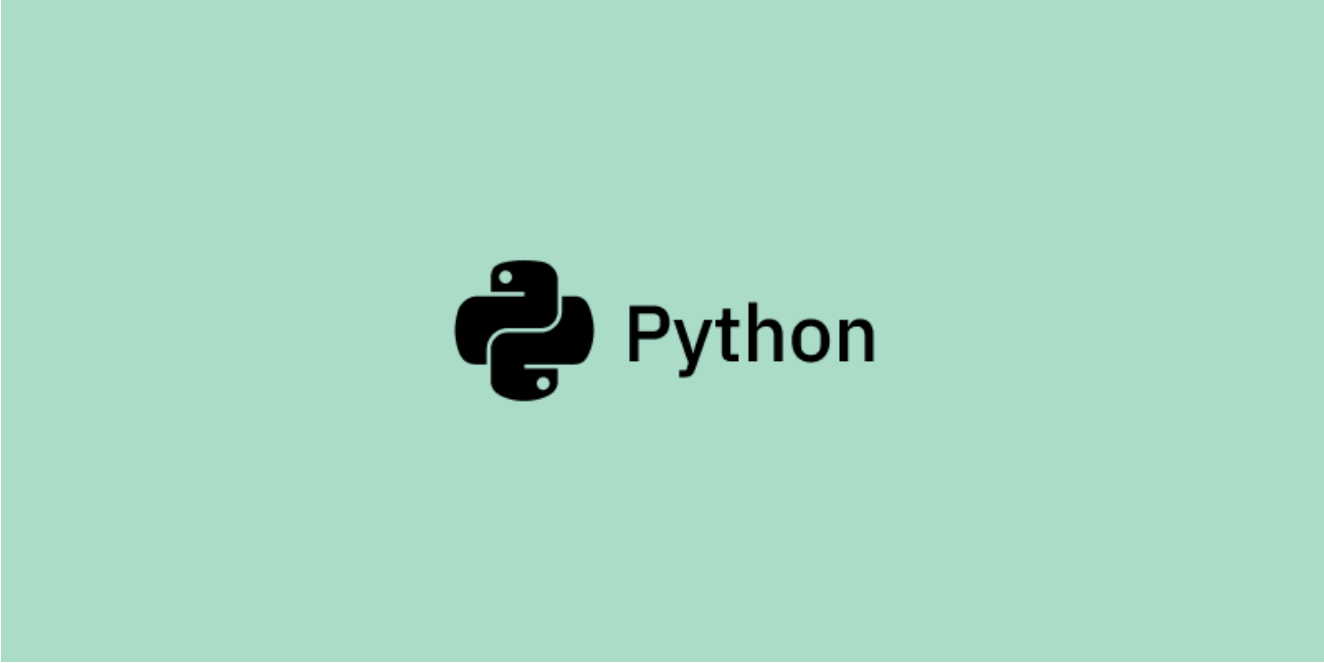
APRIL 11, 2022 / #PYTHON

# How to Set Up a Virtual Environment in Python – And Why It's Useful




By Stephen Sanwo

When developing software with Python, a basic approach is to install Python on your machine, install all your required libraries via the terminal, write all your code in a single .py file or notebook, and run your Python program in the terminal.

This is a common approach for a lot of beginners and many people transitioning from working with Python for data analytics.

multiple files, multiple packages, and dependencies. As a result, you will need to isolate your Python development environment for that particular project.

Consider this scenario: you are working on app A, using your system installed Python and you pip install packageX version 1.0 to your global Python library. Then you switch to project B on your local machine, and you install the same packageX but version 2.0, which has some breaking changes between version 1.0 and 2.0.

When you go back to run your app A, you get all sorts of errors, and your app does not run. This is a scenario you can run into when building software with Python. And to get around this, we can use virtual environments.

This tutorial will cover everything you need to know about virtual environments and how to set one up with **Virtualenv**.

# What is a Virtual Environment?

Python's official documentation says:

> *"A virtual environment is a Python environment such that the Python interpreter, libraries and scripts installed into it are isolated from those installed in other virtual environments, and (by default) any libraries installed in a "system" Python, i.e., one which is installed as part of your operating system"*

To break this down, when you activate a virtual environment for your project, your project becomes its own self contained

Your new virtual environment has its own pip to install libraries, its own libraries folder, where new libraries are added, and its own Python interpreter for the Python version you used to activate the environment.

With this new environment, your application becomes self-contained and you get some benefits such as:

- Your development environment is contained within your project, becomes isolated, and does not interfere with your system installed Python or other virtual environments
- You can create a new virtual environment for multiple Python versions
- You are able to download packages into your project without admin privileges
- You can easily package your application and share with other developers to replicate
- You can easily create a list of dependencies and sub dependencies in a file, for your project, which makes it easy for other developers to replicate and install all the dependencies used within your environment

Using virtual environments is recommended for software development projects that generally grow out of a single Python script, and Python provides multiple ways of creating and using a virtual environment.

In the sections below, we will walk through how to set up your virtual environment, using **venv**, which gives you a lot more low level control of your environment.

# How to Install a Virtual Environment using Venv

**Virtualenv** is a tool to set up your Python environments. Since Python 3.3, a subset of it has been integrated into the standard library under the venv module. You can install venv to your host Python by running this command in your terminal:

```
pip install virtualenv
```

To use venv in your project, in your terminal, create a new project folder, cd to the project folder in your terminal, and run the following command:

```
python<version> -m venv <virtual-environment-name>
```

Like so:

```
mkdir projectA
cd projectA
python3.8 -m venv env
```

When you check the new projectA folder, you will notice that a new folder called **env** has been created. env is the name of our virtual

folder. You will also see scripts that are typically used to control your virtual environment, such as activate and pip to install libraries, and the Python interpreter for the Python version you installed, and so on. (This folder will be called Scripts on windows).

The lib folder will contain a list of libraries that you have installed. If you take a look at it, you will see a list of the libraries that come by default with the virtual environment.

# How to Activate the Virtual Environment

Now that you have created the virtual environment, you will need to activate it before you can use it in your project. On a mac, to activate your virtual environment, run the code below:

```
source env/bin/activate
```

This will activate your virtual environment. Immediately, you will notice that your terminal path includes env, signifying an activated virtual environment.

Note that to activate your virtual environment on Widows, you will need to run the following code below (See this link to fully understand the differences between platforms):

```
env/Scripts/activate.bat //In CMD
```

# Is the Virtual Environment Working?

We have activated our virtual environment, now how do we confirm that our project is in fact isolated from our host Python? We can do a couple of things.

First we check the list of packages installed in our virtual environment by running the code below in the activated virtual environment. You will notice only two packages – pip and setuptools, which are the base packages that come default with a new virtual environment

```
pip list
```

Next you can run the same code above in a new terminal in which you haven't activated the virtual environment. You will notice a lot more libraries in your host Python that you may have installed in the past. These libraries are not part of your Python virtual environment until you install them.

# How to Install Libraries in a Virtual Environment

To install new libraries, you can easily just pip install the libraries. The virtual environment will make use of its own pip, so you don't need to use pip3.

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

You can name this requirements.txt file whatever you want.

# Requirements File

Why is a requirements file important to your project? Consider that you package your project in a zip file (**without the env folder**) and you share with your developer friend.

To recreate your development environment, your friend will just need to follow the above steps to activate a new virtual environment.

Instead of having to install each dependency one by one, they could just run the code below to install all your dependencies within their own copy of the project:

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

Note that it is generally not advisable to share your env folder, and it should be easily replicated in any new environment.

Typically your env directory will be included in a .gitignore file (when using version control platforms like GitHub) to ensure that the environment file is not pushed to the project repository.

# Environment

To deactivate your virtual environment, simply run the following code in the terminal:

```
~ deactivate
```

# Conclusion

Python virtual environments give you the ability to isolate your Python development projects from your system installed Python and other Python environments. This gives you full control of your project and makes it easily reproducible.

When developing applications that would generally grow out of a simple .py script or a Jupyter notebook, it's a good idea to use a virtual environment – and now you know how to set up and start using one.

---