Setting Up a Python Programming Environment

Written by Lisa Tagliaferri

Python is a flexible and versatile programming language suitable for many use cases, with strengths in scripting, automation, data analysis, machine learning, and back-end development. First published in 1991 the Python development team was inspired by the British comedy group Monty Python to make a programming language that was fun to use. Python 3 is the most current version of the language and is considered to be the future of Python.

This tutorial will help get your remote server or local computer set up with a Python 3 programming environment. If you already have Python 3 installed, along with pip and venv, feel free to move onto the next chapter!

Prerequisites

This tutorial will be based on working with a Linux or Unix-like (*nix) system and use of a command line or terminal environment. Both macOS and specifically the PowerShell program of Windows should be able to achieve similar results.

Step 1 — Installing Python 3

Many operating systems come with Python 3 already installed. You can check to see whether you have Python 3 installed by opening up a terminal window and typing the following:

```
python3 -V
```

You'll receive output in the terminal window that will let you know the version number. While this number may vary, the output will be similar to this:

Output

```
Python 3.7.2
```

If you received alternate output, you can navigate in a web browser to python.org in order to download Python 3 and install it to your machine by following the instructions.

Once you are able to type the `python3 -V` command above and receive output that states your computer's Python version number, you are ready to continue.

Step 2 — Installing pip

To manage software packages for Python, let's install pip, a tool that will install and manage programming packages we may want to use in our development projects.

If you have downloaded Python from python.org, you should have pip already installed. If you are on an Ubuntu or Debian server or computer, you can download pip by typing the following:

```
sudo apt install -y python3-pip
```

Now that you have pip installed, you can download Python packages with the following command:

```
pip3 install package_name
```

Here, `package_name` can refer to any Python package or library, such as Django for web development or NumPy for scientific computing. So if you would like to install NumPy, you can do so with the command `pip3 install numpy`.

There are a few more packages and development tools to install to ensure that we have a robust set-up for our programming environment:

```
sudo apt install build-essential libssl-dev libffi-dev python3-dev
```

Once Python is set up, and pip and other tools are installed, we can set up a virtual environment for our development projects.

Step 3 — Setting Up a Virtual Environment

Virtual environments enable you to have an isolated space on your server for Python projects, ensuring that each of your projects can have its own set of dependencies that won't disrupt any of your other projects.

Setting up a programming environment provides us with greater control over our Python projects and over how different versions of packages are handled. This is especially important when working with third-party packages.

You can set up as many Python programming environments as you want. Each environment is basically a directory or folder on your server that has a few scripts in it to make it act as an environment.

While there are a few ways to achieve a programming environment in Python, we'll be using the venv module here, which is part of the standard Python 3 library.

If you have installed Python with through the installer available from python.org, you should have venv ready to go.

To install venv into an Ubuntu or Debian server or machine, you can install it with the following:

```
sudo apt install -y python3-venv
```

With venv installed, we can now create environments. Let's either choose which directory we would like to put our Python programming environments in, or create a new directory with `mkdir`, as in:

```
mkdir environments
cd environments
```

Once you are in the directory where you would like the environments to live, you can create an environment. You should use the version of Python that is installed on your machine as the first part of the command (the output you received when typing `python -V`). If that version was `Python 3.6.3`, you can type the following:

```
python3.6 -m venv my_env
```

If, instead, your computer has `Python 3.7.3` installed, use the following command:

```
python3.7 -m venv my_env
```

Windows machines may allow you to remove the version number entirely:

```
python -m venv my_env
```

Once you run the appropriate command, you can verify that the environment is set up be continuing.

Essentially, `pyvenv` sets up a new directory that contains a few items which we can view with the `ls` command:

```
ls my_env
```

Output

```
bin include lib lib64 pyvenv.cfg share
```

Together, these files work to make sure that your projects are isolated from the broader context of your local machine, so that system files and project files don't mix. This is good practice for version control and to ensure that each of your projects has access to the particular packages that it needs. Python Wheels, a built-package format for Python that can speed up your software production by reducing the number of times you need to compile, will be in the Ubuntu 18.04 `share` directory.

To use this environment, you need to activate it, which you can achieve by typing the following command that calls the activate script:

```
source my_env/bin/activate
```

Your command prompt will now be prefixed with the name of your environment, in this case it is called my_env. Depending on what version o f Debian Linux you are running, your prefix may appear somewhat

differently, but the name of your environment in parentheses should be the first thing you see on your line:

```
(my_env) sammy@sammy:~/environments$
```

This prefix lets us know that the environment my_env is currently active, meaning that when we create programs here they will use only this particular environment's settings and packages.

Note: Within the virtual environment, you can use the command `python` instead of `python3`, and `pip` instead of `pip3` if you would prefer. If you use Python 3 on your machine outside of an environment, you will need to use the `python3` and `pip3` commands exclusively.

After following these steps, your virtual environment is ready to use.

Step 4 — Creating a "Hello, World" Program

Now that we have our virtual environment set up, let's create a traditional "Hello, World!" program. This will let us test our environment and provides us with the opportunity to become more familiar with Python if we aren't already.

To do this, we'll open up a command-line text editor such as nano and create a new file:

```
(my_env) sammy@sammy:~/environments$ nano hello.py
```

Once the text file opens up in the terminal window we'll type out our program:

```
print("Hello, World!")
```

Exit nano by typing the `CTRL` and `X` keys, and when prompted to save the file press `y`.

Once you exit out of nano and return to your shell, let's run the program:

```
(my_env) sammy@sammy:~/environments$ python hello.py
```

The `hello.py` program that you just created should cause your terminal to produce the following output:

Output

```
Hello, World!
```

To leave the environment, simply type the command `deactivate` and you will return to your original directory.

Conclusion

At this point you have a Python 3 programming environment set up on your machine and you can now begin a coding project!

If you would like to learn more about Python, you can download our free How To Code in Python 3 eBook via do.co/python-book.