05

To learn more: Virtualenv

We've already covered the importance of virtual environments in Python development, providing an overview of **venv** and how to work with it when building Python projects. However, it's crucial to mention that there is a widely used alternative called **virtualenv** , an external tool that, while serving a similar purpose, has some notable distinctions from **venv** .

Let's explore these differences and understand the circumstances under which each of these tools is most appropriate for creating virtual environments in Python projects.

Compatibility:

The `venv` is a library built into Python 3.3 and later, while `virtualenv` it is an external tool that needs to be installed separately. For newer versions of Python (3.5 and above), `venv` it is recommended as it offers similar functionality to `virtualenv` and is available by default.

Scope of Functionality:

When considering the scope of functionality, the differences between `venv` and `virtualenv` become more evident. `virtualenv` is known to be richer in functionality and supports more advanced use cases. We can see some of the significant distinctions:

Support for Different Python Interpreters:

is `virtualenv` more flexible when it comes to working with different versions of Python and alternative interpreters. It can be used to create virtual environments for multiple versions of Python, including older versions that are still compatible with `virtualenv`.

Integration with Other Languages:

A `virtualenv` is capable of integrating with languages other than Python, providing a more comprehensive solution for projects involving multiple programming languages. This can be particularly useful in complex development environments.

Isolation of Virtual Environments:

Although both `venv` provide `virtualenv` isolation of virtual environments, `virtualenv` it is known to have additional

features to deal with more complex situations, ensuring efficient separation of dependencies between different projects.

Virtual Environment Activation:

For virtualenv, you use the source `<env>/bin/activate no Linux/macOS` or command `<env>\Scripts\activate` on Windows. And for venv, the command is `source <env>/bin/activate no Linux/macOS` or `<env>\Scripts\activate` on Windows, too.

Virtualenv offers more advanced options when activating a virtual environment. In addition to the traditional commands, it allows for additional customizations, giving you greater control over the activated environment.

Python 2 Deprecation:

It is important to mention that with the end of support for Python 2, it `venv` became more viable, since `virtualenv` it was often used to create virtual environments for older versions of Python.

If you are using Python 3.3 or newer, using `venv` is recommended as it is an integrated solution. However, if you require advanced functionality or are working with older versions of Python, `virtualenv` may be a better choice.