Twitter Thread by Santiago

Santiago
@svpino

For a long time, I didn't understand how to use Virtual Environments in Python ■.

If this is just, let's end it here and now: ■■

[2] Virtual Environments let you deal with the dependencies that your code has with external Python libraries.

It avoids having conflicts when your projects depend on different versions of the same library.

■

[3] Let's imagine that you are building your first Python project and you install the "requests" library:

pip install requests

You get version 2.24.0 installed in your system.

■

[4] A month later, you decide to work on your second project. It also needs the "requests" library.

But the latest version is not 2.24.0 anymore.

Now version 3 is available, and that's the one you want to use!

■

[5] You could upgrade your entire system to version 3, but then you'll be potentially breaking the first project you built that depends on 2.24.0!

Can you imagine this happening on a server with many more applications running?

■

[6] Virtual environments solve this problem.

The first step for every new project is to create a virtual environment for it.

Some people have a central location where they store all environments. I prefer to keep them inside the project folder.

■

[7] You can create a new virtual environment with Python 3 using the following command:

python3 -m venv .myvenv

Then, you can use "source" to activate the environment.

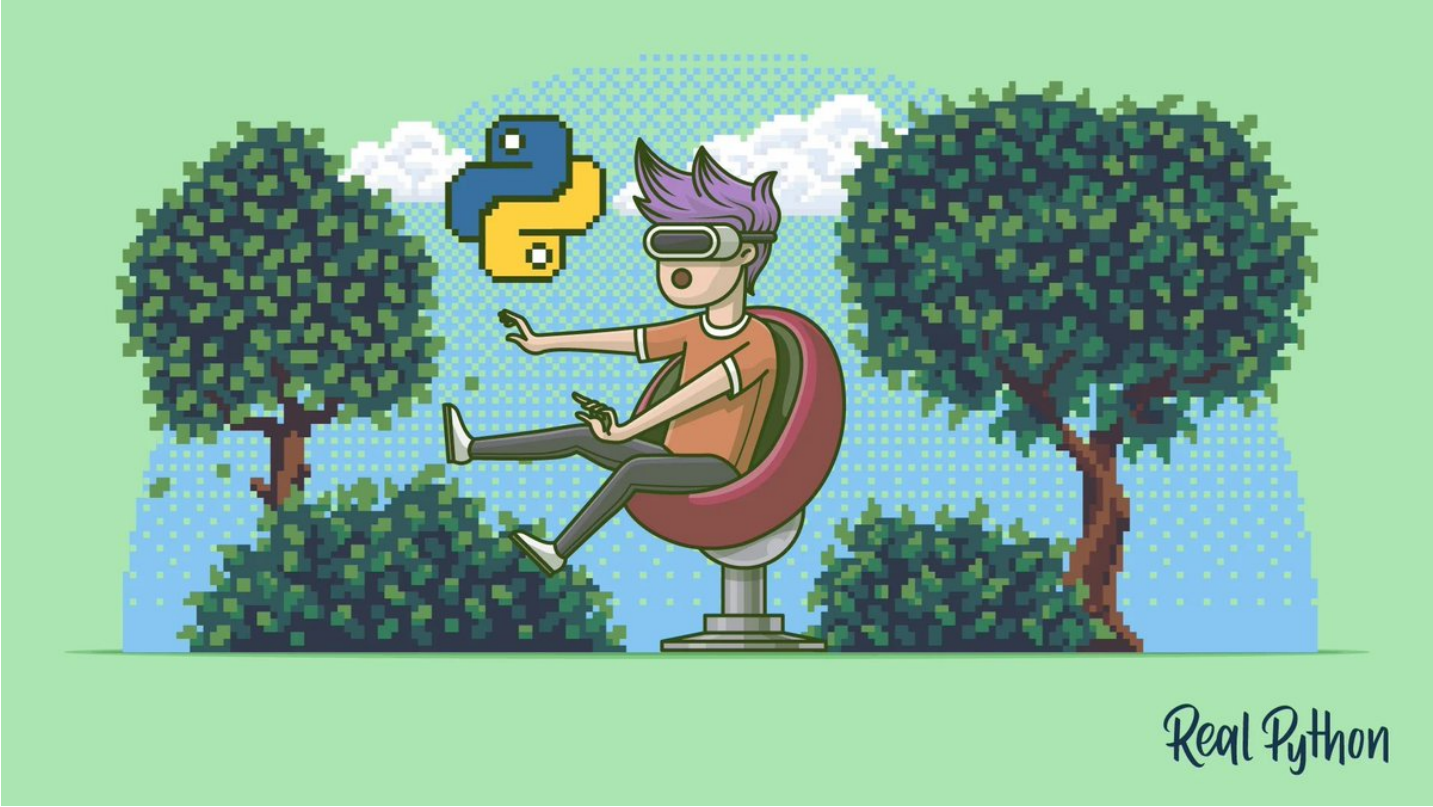
At this point, you'll have full isolation for your project.

■

[8] If you install any libraries within a virtual environment, they will never mess with the libraries installed at the system level or other virtual environments.

And this is great!

Here is a @realpython's article covering virtual environments: https://t.co/lgXqJDUlKw



[9] The built-in "venv" module is not the only way to create virtual environments. Here are other options:

- conda
- pipenv
- virtualenv

What's your choice?