Virtuelle Umgebungen

Python Projekte isolieren

Hintergrund

Jedes Python Projekt nutzt unterschiedliche Pakete und Module, die u.U. auf dem System installiert werden müssen.

Um zu verhindern, dass spezifische Versionen andere Projekte und Umgebungen beeinflussen, isoliert man diese Installationen in einer virtuellen Umgebung.

Jedes Projekt hat im Idealfall seine eigene virtuelle Umgebung.

Background

Die globale Python Installation speichert ihre Pakete alle an den gleichen Stellen. Aufgrund dieser Tatsache alleine ist es nicht möglich, mehrere unterschiedliche Versionen eines Pakets zu nutzen.

Bei einem Update eines Pakets überschreiben wir die vorliegende Version systemweit. Andere Projekte sind von diesem Vorgang unter Umständen betroffen.

Der Python Interpreter

Der Python Interpreter ist ein Programm, dass sich auf unserer Festplatte befindet. Er liegt an einem bestimmten Ort und ist über die Pfade der PATH-Variable adressierbar. Unter Windows handelt es sich um die python.exe

Wenn wir python in die Eingabeaufforderung tippen, sucht unser Betriebssystem in den Pfaden nach dem entsprechenden Programm, um es aufzurufen.

Diese Pfad-Angaben lassen sich manipulieren.

Pfad - Umgebungsvariablen

Die Pfade kann man sich unter Linux / Unix so ansehen.

echo $PATH

Unter Windows:

echo $Env:PATH

Site Packages

Falls mit pip install neue Pakete installiert werden, werden diese Pakete auf unserem System gespeichert. Den Pfad, wo diese Pakete gespeichert werden, erhalten wir mit

```
import sys
sys.prefix
```

```
import site
site.getsitepackages()
```

Virtuelle Umgebung

Um zu verhindern, dass für unser neues Projekt Pakete in den globalen Site-Packages installiert werden, müssen wir eine virtuelle Umgebung anlegen, die ihr eigenes Site-Packages-Verzeichnis hat.

Dort können diese Pakete dann abgelegt werden.

Eine virtuelle Umgebung ist also nichts weiter, als ein Verzeichnis mit einer eigenen Version des Python-Interpreters und einem eigenen site_packages-Verzeichnis.

Virtuelle Umgebungen: Tools

Es existieren mehrere konkurrierende Tools, um virtuelle Umgebungen anzulegen: venv, virtualenv, virtualenvwrapper, conda, pyenv, pipenv, uv, poetry, pyvenv sind einige davon. Wir konzentrieren uns in dem vorliegenden Dokument auf venv.

Venv ist seit Python 3.5 Teil der Standard-Library und kann ohne zusätzliche Installation genutzt werden.

Anlegen einer virtuellen Umgebung

In der Eingabeaufforderung eingeben:

python –m venv path/to/env

Das Modul venv installiert ein virtuelles Verzeichns namens env im angegebenen Pfad. Der Name env ist frei wählbar. Falls der Pfad nicht existiert, wird dieser angelegt. Z.B.:

python -m venv .envs/env

Nutzen einer virtuellen Umgebung

Um eine virtuelle Umgebung zu nutzen, muss der Code mit dem Interpreter dieser Umgebung ausgeführt werden.

Um das zum Beispiel auf der Kommandozeile zu testen, wird die virtuelle Umgebung wie folgt aktiviert:

env\Scripts\activate (windows)
source env/bin/activate (linux)

aktiviere virtuelle Umgebung in Pfad env\Scripts mit activate. Wir befinden uns nun im virtuellen Modus. Der Präfix (env) zeigt an, dass wir uns in einer virtuellen Umgebung befinden:

(env) D:\kurs\projekt\

Im virtuellen Modus pip nutzen

Wenn wir jetzt mit Pip install Pakete importieren, werden diese nur in dem Verzeichnis der genutzten virtuellen Umgebung importiert, und nicht systemweit.

(env) pip install numpy

Deaktivieren einer virtuellen Umgebung

Um aus einer virtuellen Umgebung wieder auszusteigen, nutzen wir einfach den folgenden Befehl auf der Kommandozeile:

```
deactivate
```

Damit ist der virtuelle Modus beendet.

Sys Path

In sys.path stehen die Verzeichnisse, in denen Python nach Modulen sucht. Um dynamisch Verzeichnisse hinzuzufügen, kann sys.path auch verändert werden.

```
import sys


print(sys.path)
sys.path.append(...)
```