Grundlagen zu Python

Einrichtung einer virtuellen Entwicklungsumgebung

Version: 1.5

Bodo Schönfeld

8. November 2023

Inhaltsverzeichnis

1 Einführung

Was macht eine virtuelle Entwicklungsumgebung? Sie isoliert verschiedene Python-Versionen und deren Paket-Installationen voneinander. Einmal für ein Python-Projekt eingerichtet, bedeutet dies, dass die verwendeten Pakete nur diesem Projekt zur Verfügung stehen. Die Aktualisierung eines dieser Pakete beeinflusst *nicht* andere — auf dem System installierte — Pakete. Darüber hinaus können für unterschiedliche Projekte unterschiedliche Python-Versionen installiert und verwendet werden.

Pakete nicht global zu installieren hat einen weiteren Vorteil: Es schützt die Integrität des Betriebssystems. Außerdem werden Paket-Konflikte vermieden, die bei einer systemweiten Installation auftreten können.

2 virtualenv und venv

Um eine virtuelle Entwicklungsumgebung verwenden zu können, war es früher notwendig, das Paket **virtualenv** zu installieren. Seit Version 3.4 ist dies nicht mehr erforderlich, da Teile dieses Pakets als Modul mit dem Namen **venv** in Python implementiert wurden. Die Verwendung von `venv` ist seit Version 3.5 der empfohlene Weg zur Einrichtung einer virtuellen Umgebung.[1] Deswegen wird nachfolgend die Einrichtung anhand dieses Moduls erklärt.

3 Eine virtuelle Umgebung einrichten

Nachdem ein neues Projekt eingerichtet wurde, kann auch schon die virtuelle Umgebung hinzugefügt werden. Betrachten wir dies an folgendem Beispiel. Angenommen das Python-Projekt trägt den Namen „my-project“, dann könnte die Projektstruktur wie folgt aussehen:[2]

```
my-project
├── README.md
├── pytest.ini
├── requirements.txt
├── setup.cfg
├── setup.py
├── docs
├── my_project
│   ├── __init__.py
│   └── main.py
└── tests
    └── __init__.py
```

Sofern noch nicht geschehen, wechselt man zunächst im Terminal oder der PowerShell mit dem Befehl `cd` in das Projektverzeichnis:

[1] Vgl. die Dokumentation zu venv: https://docs.python.org/3/library/venv.html
[2] Artikel zur Struktur eines Python-Projekts: https://bodo-schoenfeld.de/projektstruktur-python/

```
$ cd my-project   # Linux, macOS
PS> cd my-project # Windows
```

Die virtuelle Umgebung wird jetzt mit folgender Anweisung erstellt:

```
$ python3 -m venv venv  # Linux, macOS
PS> python -m venv venv # Windows
```

Unter **Windows** ist es möglich, anstelle von `python` lediglich `py` zu verwenden:

```
PS> py -m venv venv
```

Hinweis: Falls unter Windows der Pfad zur Python-Installation nicht zur PATH-Variable hinzugefügt wurde, muss der vollständige Pfad für das Erstellen einer virtuellen Umgebung verwendet werden.

Die Ausführung dieser Anweisung führt dazu, dass ein neues Verzeichnis angelegt wird. In diesem Beispiel hat es die Bezeichnung **venv**. Der Name des Verzeichnisses ist frei wählbar, allerdings ist es unter Python-Entwicklern allgemein üblich „venv“ (oder „.venv“) zu verwenden. Insbesondere auf Linux-Systemen und unter macOS wird häufig vor dem Verzeichnisnamen einen Punkt gesetzt, also „.venv“ verwendet.

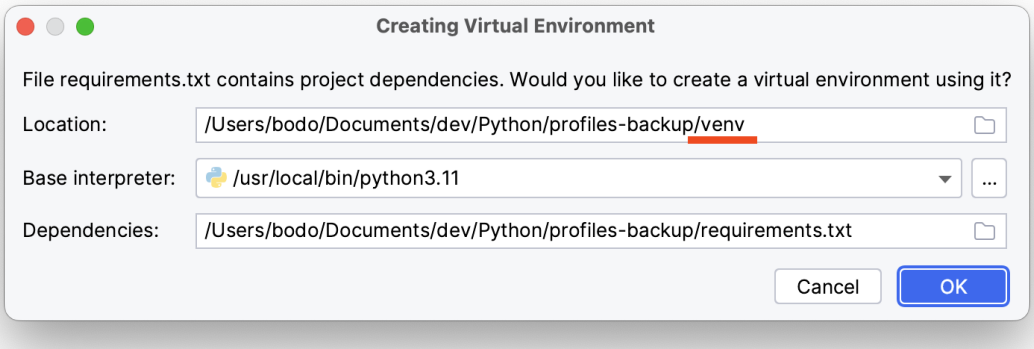
```
$ python3 -m venv .venv
```

Durch die Verwendung von „.venv“ wird dieses Ordner zu einem versteckten Ordner, der nicht mehr im Datei-Manager von Linux oder macOS angezeigt wird.

Wird eine virtuelle Umgebung über eine Entwicklungsumgebung erstellt, wird in der Regel ebenfalls ein Ordner mit der Bezeichnung „venv“ oder „.venv“ angelegt. So erzeugt **Visual Studio Code** das Verzeichnis „.venv“, während **PyCharm** als Verzeichnisname „venv“ vorschlägt.[3]

[3] Mit der Bezeichnung „.venv“ hat PyCharm aber ebenfalls keine Probleme.

Abbildung 1: PyCharm möchte eine virtuelle Umgebung erstellen



Microsoft empfiehlt übrigens in der Dokumentation zur Erstellung virtueller Umgebungen die Bezeichnung „.venv“[4].

Es ist nicht ungewöhnlich, dass auf einem Rechner mehrere Python-Versionen installiert sind. In diesem Fall besteht die Möglichkeit, die zu verwendende Python-Version anzugeben:

```
$ python3.12 -m venv .venv
```

Nachdem eine virtuelle Umgebung erzeugt wurde, sieht die Verzeichnisstruktur wie folgt aus, wobei hier als Ordnername „.venv“ gewählt wurde:

```
my-project
├── .venv
│   ├── bin
│   ├── include
│   ├── lib
│   └── pyvenv.cfg
├── README.md
└── pytest.ini
```

[4]Dokumentation zu Visual Studio Code: https://code.visualstudio.com/docs/python/environments

```
├── requirements.txt
├── setup.cfg
├── setup.py
├── docs
├── my_project
│   ├── __init__.py
│   └── main.py
└── tests
    └── __init__.py
```

4 Eine virtuelle Umgebung aktivieren oder deaktivieren

Bevor jetzt Pakete für dieses Projekt installiert werden können, muss die virtuelle Umgebung aktiviert werden. Die nach der Aktivierung installierten Pakete stehen dann nur diesem Projekt (und *nicht* global) zur Verfügung.

Unter **Linux** oder **macOS** ist folgende Anweisung zu verwenden:

```
$ source .venv/bin/activate
```

Und unter **Windows** ist folgende Anweisung zu verwenden:

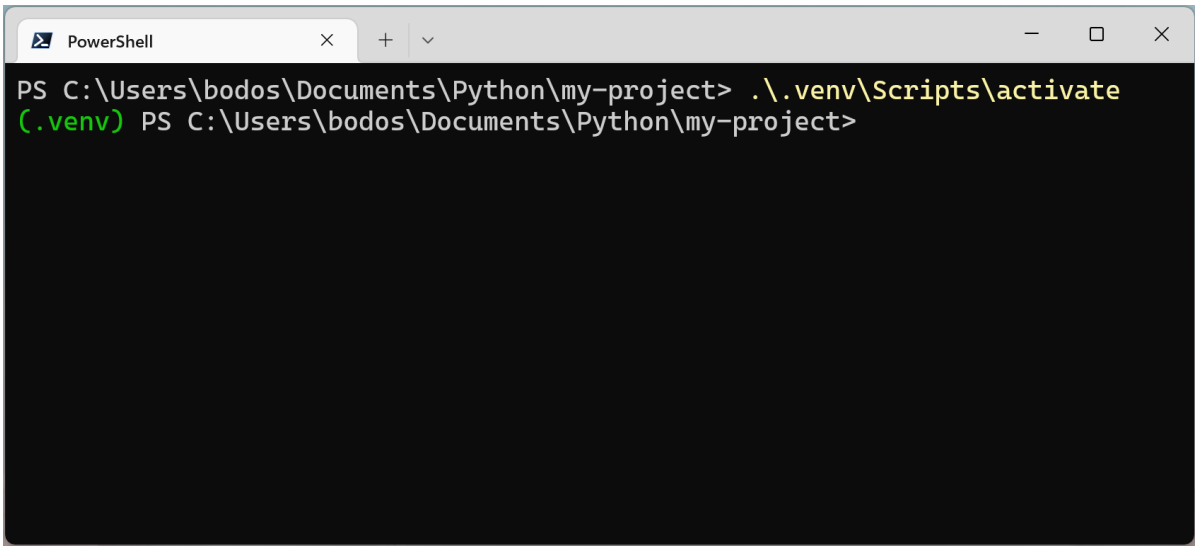
```
PS> .venv\Scripts\activate
```

Nach der Aktivierung wird die Eingabeaufforderung am Anfang der Zeile um (`.venv`) ergänzt.

Nach der Aktivierung können mit `pip` bzw. `pip3` Pakete installiert werden:

```
pip3 install <Paketname>  # Linux, macOS
pip install <Paketname>   # Windows
```

Abbildung 2: Virtuelle Umgebung in der PowerShell



Wie bereits erwähnt, stehen diese Pakete nur in diesem Projekt zur Verfügung. Dadurch wird sichergestellt, dass für verschiedene Projekte unterschiedliche Versionen installiert werden können.

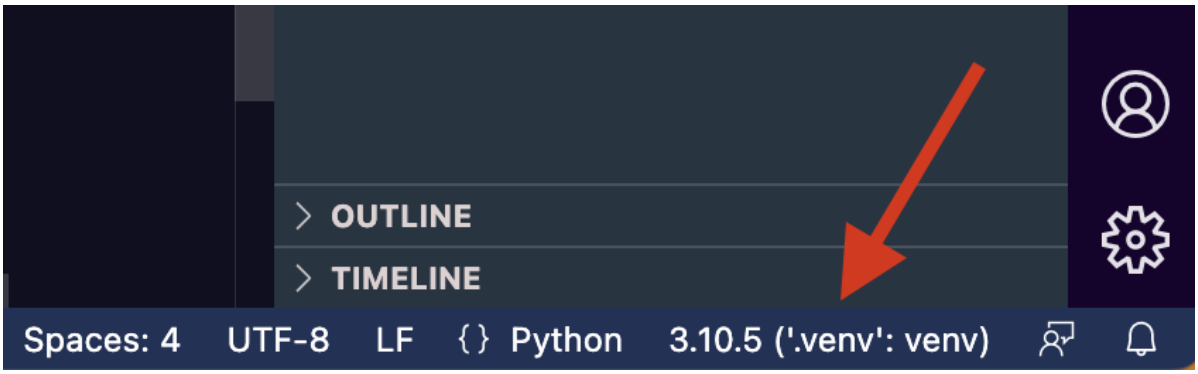
Die **Deaktivierung** der virtuellen Umgebung erfolgt auf allen Betriebssystemen mit dem Befehl `deactivate`.

5 Entwicklungsumgebungen

Eine im Terminal oder der PowerShell erstellte virtuelle Umgebung wird von Entwicklungsumgebungen oder Code Editoren grundsätzlich automatisch erkannt. Im allgemeinen kann man sagen, dass dies immer dann der Fall ist, wenn als Verzeichnisname „.venv“ oder „venv“ verwendet wird. Eine erkannte Entwicklungsumgebung muss also nicht von Hand aktiviert werden. Dies sollte bereits von der Entwicklungsumgebung erledigt worden sein.

Überprüfen lässt sich dies in der Statusleiste der jeweiligen Anwendung. Dort steht dann beispielsweise „(.venv)“. In manchen Programmen wird zusätzlich auch der verwendete Python-Interpreter angezeigt.

Abbildung 3: Hinweis zur aktivierten virtuellen Umgebung in VS Code



Sollte **Visual Studio Code** eine vorhanden virtuelle Umgebung nicht automatisch erkennen, kann sie über die **Command Palette** aktiviert werden. Gebt dazu als Suchbegriff „env“ ein. Hierüber kann auch eine neue virtuelle Umgebung erstellt werden, wobei Visual Studio Code standardmäßig als Verzeichnisnamen „.venv“ wählt.

Nutzer von **PyCharm** erhalten bei nicht vorhandener virtueller Umgebung die Möglichkeit, einen Python-Interpreter auszuwählen. Optional kann dabei auch eine virtuelle Umgebung erzeugt werden, wobei standardmäßig als Verzeichnisname „venv“ vorgeschlagen wird.

6 Versionskontrolle (Git)

Es ist noch zu erwähnen, dass das Verzeichnis „.venv“ (oder „venv“) nicht in eine Versionskontrolle (git) aufgenommen werden sollte. Denn dies würde dazu führen, dass die für dieses Projekt installierten Pakete ebenfalls zu Github (oder einem anderen Anbieter) übertragen würden.

Stattdessen bietet es sich an, zu dem Projekt die Datei „requirements.txt“ hinzuzufügen. In dieser Datei werden alle zum Projekt gehörenden Module aufgeführt, so dass sie bei Bedarf mit dem Befehl

```
$ pip3 install -r requirements.txt  // Linux, macOS
PS> pip install -r requirements.txt // Windows
```

installiert werden können.

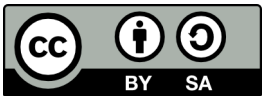
Ein Ausschluss des Ordners „.venv“ (oder „venv“) wird durch einen entsprechenden Eintrag in der Datei „.gitignore“ erreicht. In dieser Datei muss einfach die Zeile „/.venv“ hinzugefügt werden. Eine typische .gitignore-Datei könnten folgenden Inhalt haben:

```
/.idea
/.vscode
/.venv
/build
/dist
/docs
```

7 Lizenz

- Version: 1.5 vom 8. November 2023
- Autor: @niftycode
- Homepage: https://bodo-schoenfeld.de
- Github: https://github.com/niftycode

Creative Commons Lizenz: Namensnennung — Weitergabe unter gleichen Bedingungen 4.0 International (CC BY-SA 4.0):



- **Teilen** — das Material in jedwedem Format oder Medium vervielfältigen und weiterverbreiten.
- **Bearbeiten** — das Material remixen, verändern und darauf aufbauen und zwar für beliebige Zwecke, sogar kommerziell.