Grundlagen zu Python

Einrichtung einer virtuellen Entwicklungsumgebung

Version: 1.9

Bodo Schönfeld

9. Oktober 2024

Inhaltsverzeichnis

1 Einführung 4

2 virtualenv und venv 5
2.1 macOS und Windows 5
2.2 Linux 5

3 Eine virtuelle Umgebung einrichten 6

4 Eine virtuelle Umgebung aktivieren oder deaktivieren 10

5 Entwicklungsumgebungen 12
5.1 Visual Studio Code 12
5.2 PyCharm 13
5.3 Zed 13

6 Versionskontrolle (Git) 15

7 Lizenz 16

Im Gegensatz zu den vorherigen Versionen, können etwaige Links, die in Fußnoten vorkommen, nunmehr angeklickt werden. Darüber hinaus wurde das Kapitel „Entwicklungsumgebungen“ erweitert. Denn seit einiger Zeit wird der (in Rust geschriebene) Code-Editor Zed (`https://zed.dev`) immer beliebter. Daher habe ich zu dieser Version einen Abschnitt hinzugefügt, in dem die Aktivierung einer virtuellen Umgebung in Zed erklärt wird. Darüber hinaus wird nun in einem separaten Abschnitt erklärt, wie das Paket venv auf Linux-Distributionen, die auf Debian basieren, installiert werden kann.

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.

2.1 macOS und Windows

Sobald unter macOS oder einem Windows-System Python installiert wurde, steht das Modul **venv** zur Verfügung.

2.2 Linux

Auf Linux-Systemen muss nach der Installation von Python zusätzlich das entsprechende **venv**-Paket installiert werden. Um herauszufinden, wie der genaue Name des Pakets ist, bietet es sich an, im Paketmanager nach „venv“ zu suchen. Auf Linux-Distributionen, die auf **Debian** basieren, kann eine Installation für die Python-Version 3.11 mit folgender Anweisung durchgeführt werden:

```
$ sudo apt install python3.11-venv
```

[1] Vgl. die Dokumentation zu venv: `https://docs.python.org/3/library/venv.html`

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:[1]

```
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:

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

[1] Artikel zur Struktur eines Python-Projekts: `https://bodo-schoenfeld.de/projektstruktur-python/`

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.[2]

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 dieser Ordner zu einem versteckten Ordner, der nicht mehr im Datei-Manager von Linux oder macOS angezeigt wird.

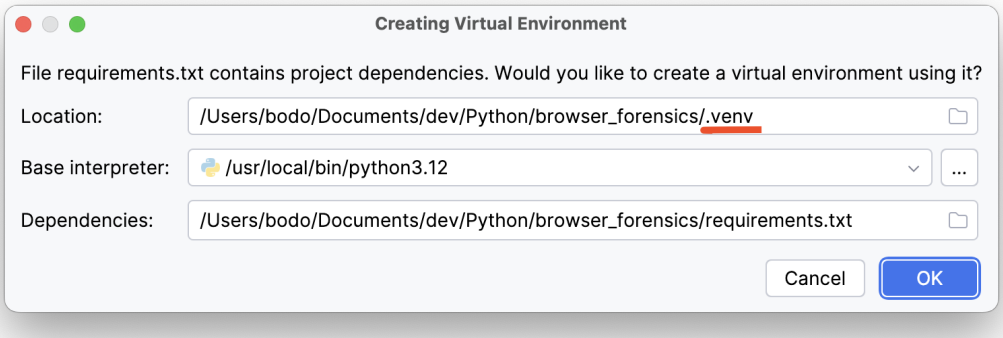
Wird eine virtuelle Umgebung über eine Entwicklungsumgebung erstellt, wird in der Regel ebenfalls ein Ordner mit der Bezeichnung „venv“ oder „.venv“ angelegt. So erstellen **Visual Studio Code** und **PyCharm** das Verzeichnis „.venv“.

Microsoft empfiehlt übrigens in der Dokumentation zur Erstellung virtueller Umgebungen die Bezeichnung „.venv“[3].

[2] Das Hinzufügen des Pfades zur PATH-Variable kann während des Installationsprozesses von Python vorgenommen werden. Wie Python auf einem Windows-System installiert werden kann, erklärt dieser Artikel: `https://bodo-schoenfeld.de/installation-von-python-unter-windows-10/` Darüber hinaus ist es aber auch möglich, die PATH-Variable nachträglich zu bearbeiten. Wie Umgebungsvariablen bearbeitet werden können, wird in diesem Artikel erklärt: `https://bodo-schoenfeld.de/umgebungsvariablen-in-windows-11-bearbeiten/`

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

Abbildung 3.1: PyCharm möchte eine virtuelle Umgebung erstellen

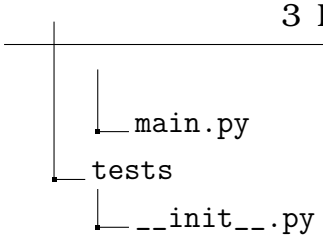


Es ist nicht ungewöhnlich, dass auf einem Rechner mehrere Python-Versionen installiert sind. In diesem Fall besteht die Möglichkeit, die zu verwendende Python-Version anzugeben:

```
$ 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
├── 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.

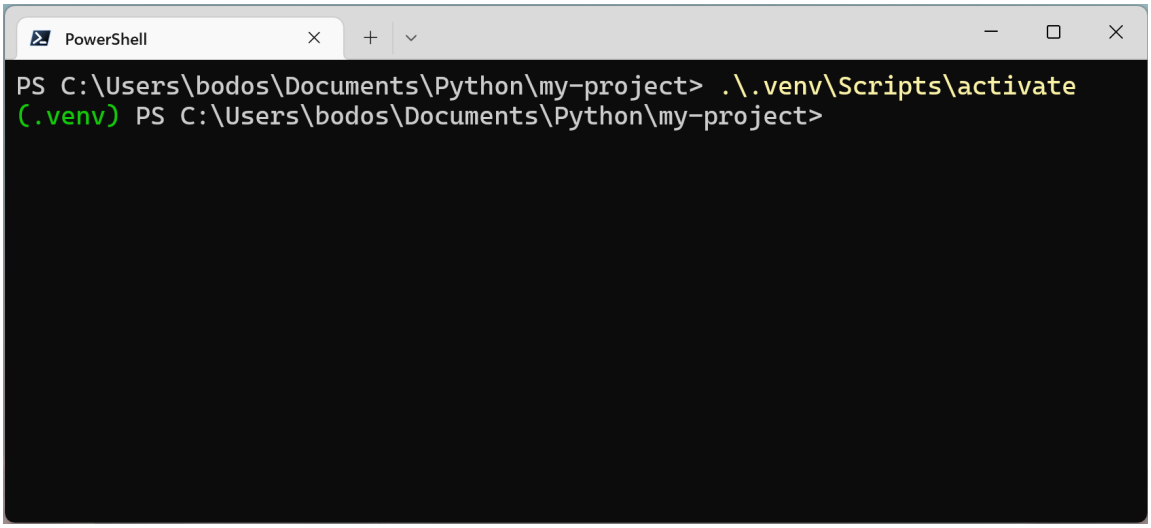
Nach der Aktivierung können mit `pip` bzw. `pip3` Pakete installiert werden:

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

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`.

Abbildung 4.1: Virtuelle Umgebung in der PowerShell

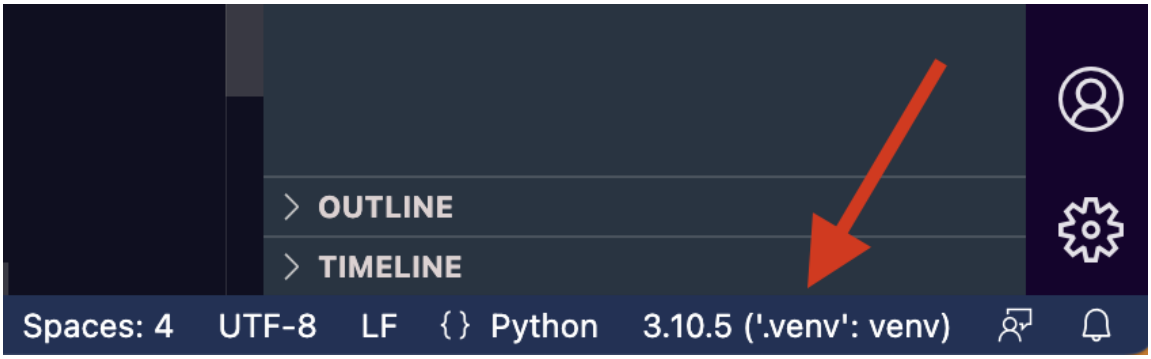


5 Entwicklungsumgebungen

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 manuell 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 5.1: Hinweis zur aktivierten virtuellen Umgebung in VS Code

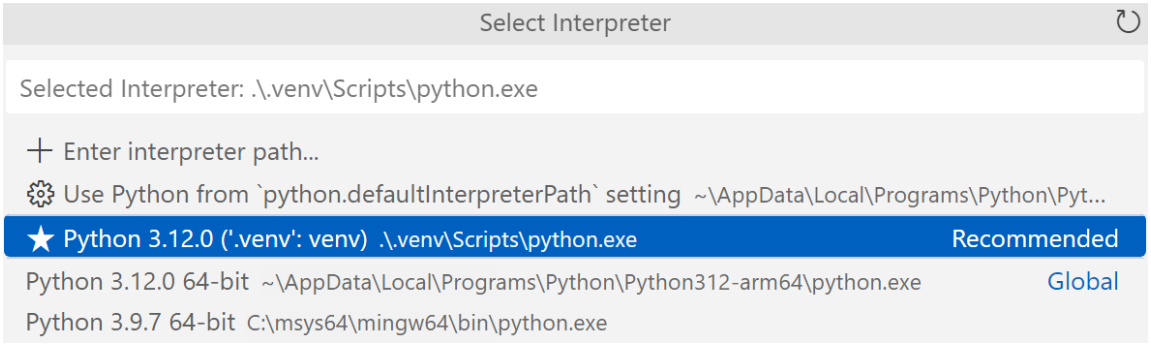


5.1 Visual Studio Code

Sollte **Visual Studio Code** eine vorhanden virtuelle Umgebung nicht automatisch erkennen, kann sie über die **Command Palette** (Windows/Linux: `Shift` + `Strg` +

P | macOS: Shift + Command + P) 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“ verwendet. Falls auf dem System mehrere Python-Interpreter zur Verfügung stehen, werden diese zur Auswahl angeboten.

Abbildung 5.2: Python-Interpreter in VS Code auswählen



5.2 PyCharm

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.

5.3 Zed

Damit eine virtuelle Umgebung verwendet werden kann, ist es erforderlich zum Projekt eine JSON-Datei mit der Bezeichnung „pyrightconfig.json“ hinzufügen. Diese Datei hat folgenden Inhalt:

```
{
  "venvPath": ".",
  "venv": ".venv"
```

```
}
```

Alternativ kann auch eine TOML-Datei verwendet werden, die die Bezeichnung „pyproject.toml“ haben muss. Diese Datei hat folgenden Inhalt:

```
{
  [tool.pyright]
  venvPath = "."
  venv = ".venv"
}
```

Wie den Beispielen entnommen werden kann, wird hier davon ausgegangen, dass das Verzeichnis für die virtuelle Umgebung den Namen „.venv“ hat.

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.

Ein Ausschluss von „.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
```

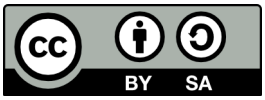
Es ist sinnvoll zum Projekt die Datei „requirements.txt“ hinzuzufügen. In diese Datei werden alle zum Projekt gehörenden Module aufgeführt. Nachdem ein Projekt geklont und die virtuelle Umgebung aktiviert wurde, können die Module dann mit folgendem Befehl installiert werden:

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

7 Lizenz

- Version: 1.9
- 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.