`pip`

`pip` ist ein Paketverwaltungstool für Python, das es Entwicklern ermöglicht, externe Bibliotheken und Pakete zu installieren, zu aktualisieren und zu verwalten. Es ist standardmäßig in Python-Distributionen ab Version 3.4 enthalten und wird häufig verwendet, um die Abhängigkeiten eines Projekts zu verwalten.

Wichtige Befehle von Pip

- **Installation von Paketen**:
  ```
  pip install <paketname>
  ```
  ```
  pip install numpy
  ```
- **Installation einer bestimmten Version eines Pakets**:
  ```
  pip install <paketname>==<version>
  ```
  ```
  pip install pandas==1.2.0
  ```
- **Aktualisierung eines Pakets**:
  ```
  pip install --upgrade <paketname>
  ```
  ```
  pip install --upgrade requests
  ```
- **Deinstallation eines Pakets**
  ```
  pip uninstall <paketname>
  ```
  ```
  pip uninstall matplotlib
  ```
- **Auflisten aller installierten Pakete**:
  ```
  pip list
  ```
- **Exportieren installierter Pakete in eine Datei**:
  ```
  pip freeze > requirements.txt
  ```
- **Installieren von Paketen aus einer Datei**:
  ```
  pip install -r requirements.txt
  ```
- **Überprüfen der Version von pip**:
  ```
  pip --version
  ```
- **Suchen nach Paketen**:
  ```
  pip search <suchbegriff>
  ```

Python´s virtuelle Umgebung `venv`

`venv` ist ein Modul in Python, das verwendet wird, um virtuelle Umgebungen zu erstellen. Virtuelle Umgebungen sind isolierte Arbeitsumgebungen, in denen Python-

Projekte unabhängig von den globalen Python-Installationen und deren Paketen entwickelt werden können. Hier sind einige wichtige Punkte zu `venv`:

Vorteile von `venv`:

- **Isolierung**: Jedes Projekt hat seine eigenen Abhängigkeiten, sodass unterschiedliche Projekte unterschiedliche Versionen von Paketen verwenden können, ohne Konflikte zu verursachen.
- **Einfachheit**: `venv` ist Teil der Standardbibliothek in Python 3, sodass es keine zusätzlichen Installationen benötigt.
- **Portabilität**: Virtuelle Umgebungen können einfach in verschiedene Umgebungen kopiert werden, was die Zusammenarbeit und den Deployment-Prozess erleichtert.

In der aktivierten Umgebung können Pakete mit `pip` installiert werden, **ohne die globale Python-Installation zu beeinflussen**.

- **Erstellen einer virtuellen Umgebung**:

  ```
  python -m venv .my_venv
  ```

  Hierbei wird eine neue virtuelle Umgebung mit dem Namen `.my_venv` erstellt.

- **Aktivieren der virtuellen Umgebung**:

  - **Windows**:
    `.myenv\Scripts\activate.bat`

  - **Linux/macOS**:

    ```
    source .myenv/bin/activate
    ```

- **Deaktivieren der virtuellen Umgebung**:

  ```
  .myenv\Scripts\deactivate.bat
  ```

Optionen für `python -m venv`:

1. **`-p / --python`**
   Erlaubt die Version des Python Interpreters anzugeben.
   `python -m venv myenv --python=python3.8`

Anzeigen aller installierten Python Versionen (geht so nur unter Windows):

```
py -0
```

2. **`--system-site-packages`** :

   Erlaubt der virtuellen Umgebung, auf die global installierten Pakete (im System-Python) zuzugreifen. Wenn diese Option nicht gesetzt ist, hat die venv keinen Zugriff auf systemweite Pakete.

   ```
   python -m venv myenv --system-site-packages
   ```

3. **`--symlinks / --copies`** :

   - **`--symlinks`** (Standard auf unterstützten Plattformen): Erstellt symbolische Links zu den Python-Binärdateien, anstatt Kopien anzulegen. Dies spart Speicherplatz.

   - **`--copies`** : Erstellt Kopien der Python-Binärdateien, anstatt symbolische Links zu verwenden. Dies kann auf Systemen nützlich sein, die keine symbolischen Links unterstützen.

     ```
     python -m venv myenv --copies
     ```

4. **`--clear`** :

   Löscht den Inhalt des venv-Verzeichnisses, wenn es bereits existiert, bevor es neu erstellt wird.

   ```
   python -m venv myenv --clear
   ```

5. **`--upgrade`** :

   Aktualisiert eine vorhandene virtuelle Umgebung, falls neue Versionen von Python installiert wurden.

   ```
   `python -m venv myenv --upgrade`
   ```

6. **`--upgrade-deps`** :

   Aktualisiert automatisch `pip` und `setuptools` in der neu erstellten virtuellen Umgebung.

   ```
   `python -m venv myenv --upgrade-deps`
   ```

7. **`--without-pip`** :

   Erstellt eine virtuelle Umgebung ohne `pip` . Normalerweise wird `pip` standardmäßig in einer neuen Umgebung installiert.

   `python -m venv myenv --without-pip`

8. **`--prompt`** :

   Ändert den Namen, der in der Kommandozeilen-Eingabeaufforderung angezeigt wird, wenn du in der virtuellen Umgebung arbeitest.
   `python -m venv myenv --prompt "my-custom-prompt"`

Bibliotheken der virtuellen Umgebung laden und speichern

Die `pip` Anweisung **"freeze"** ermöglicht die aktuell installierten Python-Pakete in einer virtuellen Umgebung in eine Datei zu exportieren. Diese Datei kann dann verwendet werden, um dieselben Pakete in einer anderen Umgebung zu installieren.

- **Pakete auflisten und in eine Datei schreiben**:
  `pip freeze > requirements.txt`

- **Installieren von Paketen aus der `requirements.txt` -Datei**:
  `pip install -r requirements.txt`

Python cmd

- **`python`** :

Dies ist der direkte Befehl, um den Python-Interpreter aufzurufen.
In der Kommandozeile gibt der Befehl `python` an, dass der Python-Interpreter gestartet werden soll.
Auf manchen Windows-Systemen kann es sein, dass der Befehl `python` nicht verfügbar ist oder falsch verknüpft ist, da Windows standardmäßig keine Unix-ähnlichen Befehle nutzt.

`python my_script.py`

- **`py`** :

Dies ist ein Python-Launcher für Windows, der speziell entwickelt wurde, um die Python-Versionen zu verwalten und Python-Skripte einfach auszuführen.
Der Befehl `py` löst den Python-Launcher auf Windows aus. Es bietet einige Vorteile, wie das einfache Starten verschiedener Python-Versionen (z. B. Python 2 oder Python 3), ohne dass die Umgebungsvariablen manuell angepasst werden müssen.
Du kannst direkt angeben, welche Python-Version verwendet werden soll, indem du nach dem `py` -Befehl die Versionsnummer angibst (z. B. `py -2` für Python 2 oder `py -3` für Python 3).