Erstellen eigener Python-Module und -Pakete

Python ermöglicht es Ihnen, eigenen Code in **Modulen** und **Paketen** zu organisieren. Dies hilft, den Code wiederverwendbar, strukturiert und gut organisiert zu halten. Hier ist eine Anleitung, wie Sie Ihre eigenen Module und Pakete erstellen und verwalten.

1. Was ist ein Modul?

Ein **Modul** ist eine einzelne Python-Datei (`.py`), die Funktionen, Klassen und Variablen enthält, die in anderen Teilen des Programms wiederverwendet werden können.

Beispiel: Erstellen eines Moduls

Erstellen Sie eine Datei namens `mein_modul.py` und definieren Sie Funktionen oder Variablen darin:

```
# mein_modul.py

def hallo():
    return "Hallo, Welt!"

variable = 42
```

Nun können Sie dieses Modul in anderen Python-Dateien importieren:

```
# main.py

import mein_modul

print(mein_modul.hallo())   # Ausgabe: Hallo, Welt!
print(mein_modul.variable)   # Ausgabe: 42
```

2. Was ist ein Paket?

Ein **Paket** ist eine Sammlung von Modulen in einem Verzeichnis, das eine spezielle `__init__.py`-Datei enthält. Diese Datei macht das Verzeichnis zu einem Paket und ermöglicht das Importieren von Modulen innerhalb des Pakets.

Struktur eines einfachen Pakets

Angenommen, wir möchten ein Paket namens `mein_paket` erstellen, das mehrere Module enthält. Die Struktur könnte wie folgt aussehen:

```
mein_paket/
│
├── __init__.py
```

```
├── modul1.py
└── modul2.py
```

Beispiel: Erstellen eines Pakets

1. **Verzeichnis und `__init__.py` erstellen**
   Erstellen Sie das Verzeichnis `mein_paket` und eine leere `__init__.py`-Datei. Die `__init__.py` kann auch Initialisierungscode enthalten, wird aber oft leer gelassen.

2. **Module im Paket erstellen**
   Erstellen Sie beispielsweise zwei Module `modul1.py` und `modul2.py`:

   ```python
   # mein_paket/modul1.py
   def funktion1():
       return "Dies ist Funktion 1 in Modul 1"

   # mein_paket/modul2.py
   def funktion2():
       return "Dies ist Funktion 2 in Modul 2"
   ```

3. **Paket importieren und verwenden**
   Sie können nun das Paket und seine Module in einer anderen Datei importieren:

   ```python
   # main.py

   from mein_paket import modul1, modul2

   print(modul1.funktion1())  # Ausgabe: Dies ist Funktion 1 in Modul 1
   print(modul2.funktion2())  # Ausgabe: Dies ist Funktion 2 in Modul 2
   ```

   Alternativ können Sie auch `from mein_paket.modul1 import funktion1` verwenden, um nur eine bestimmte Funktion zu importieren. Dies bietet allerdings keinen Gewinn hinsichtlich Speicherverbrauch. Auch wenn nur eine Funktion importiert wird, wird das gesamte Modul geladen. Vorteil ist hier lediglich der Zugriff auf die funktion ohne den Modul-Nampespace.

3. Erstellen eines installierbaren Python-Pakets

Wenn Sie Ihr Paket als installierbares Python-Paket verteilen möchten, müssen Sie zusätzliche Dateien hinzufügen. Die gängigste Methode war lange Zeit die Nutzung von `setuptools` und einer `setup.py`-Datei.

Heute wird eine pyproject.toml Datei bevorzugt.

Projektstruktur für ein installierbares Paket

```
mein_projekt/
│
├── mein_paket/
│   ├── __init__.py
│   ├── modul1.py
│   └── modul2.py
│
├── setup.py
└── README.md
```

Inhalt der `setup.py`

Die `setup.py` ist eine Konfigurationsdatei für `setuptools`, die die Metadaten des Pakets und Installationsanweisungen enthält:

```
# pyroject.toml

[build-system]
requires = ["setuptools"]
build-backend = "setuptools.build_meta"

[project]
name = "my-project"
version = "1.2.3"
```

Installation des Pakets

Nach dem Erstellen der `setup.py`-Datei können Sie das Paket lokal installieren, um es in anderen Projekten zu verwenden:

```
pip install -e .
```

Das `-e`-Flag installiert das Paket im "Entwicklungsmodus", sodass Änderungen an den Dateien sofort verfügbar sind.

4. Veröffentlichung eines Pakets auf PyPI

Falls Sie Ihr Paket öffentlich zugänglich machen möchten, können Sie es im Python Package Index (PyPI) veröffentlichen. Dies erfordert zusätzliche Schritte:

1. **Erstellen eines PyPI-Accounts**: Registrieren Sie sich auf PyPI.
2. **Erstellen des Pakets**: Erstellen Sie das Paket mit `setuptools` und `twine`.

```
python setup.py sdist bdist_wheel
```

3. **Veröffentlichen mit Twine**:

```
pip install twine
twine upload dist/*
```

Nach dem Hochladen ist Ihr Paket für andere Entwickler über `pip install mein_paket` verfügbar.