Ein lokal installierbares Python Package erstellen

Verzeichnis-Struktur

```
testpro_project
  ├── pyproject.toml
  ├── README.md
  └── testpro
      ├── __main__.py
      └── prodder.py
```

Distutils

Distutils ist eine ältere Python-Bibliothek, die das Erstellen, Verteilen und Installieren von Python-Paketen ermöglicht, jedoch mittlerweile weitgehend durch modernere Tools wie setuptools ersetzt wurde.

```
https://docs.python.org/3.10/library/distutils.html
```

Deprecated für python 3.12

Setuptools Website

Setuptools ist eine Sammlung von Erweiterungen der Python-distutils, die es Entwicklern ermöglicht, Python-Pakete – insbesondere solche mit Abhängigkeiten zu anderen Paketen – einfacher zu erstellen und zu verteilen. Pakete, die mithilfe von setuptools erstellt und verteilt werden, wirken für Anwender wie ganz normale, auf den distutils basierende Python-Pakete.

https://setuptools.pypa.io/en/latest/setuptools.html

main.py File

Im Kontext des Python-Packagings ist die Datei **main**.py innerhalb eines Pakets die Stelle, an der der Python-Interpreter ansetzt, wenn das Paket als Skript ausgeführt wird (z. B. mit dem Befehl python -m deinpaket). Dadurch dient **main**.py als „Einstiegspunkt“ und definiert, was passieren soll, wenn man das Paket selbst als ausführbares Programm verwendet.

```
python -m testpro
```

später, wenn wir das Package installiert haben, können wir es auch direkt ausführen

```
python testpro
```

das Pyproject.toml File

pyproject.toml ist eine zentrale Konfigurationsdatei in Python-Projekten, in der Abhängigkeiten, Build-Einstellungen und andere Metadaten festgelegt werden.

```
https://setuptools.pypa.io/en/latest/userguide/pyproject_config.html


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

[project]
  name = "testpro"
  version = "0.0.1"
  authors = [
      {name = "Knoto", email = "knoto@example.com"},
  ]
  description = "The testpro description"
  readme = "README.md"
  requires-python = ">=3.11"
  license = {text = "BSD-3-Clause"}
  classifiers = [
      "Programming Language :: Python :: 3",
  ]
  dependencies = [
      "numpy",
      'importlib-metadata; python_version<"3.12"',
  ]
```

Wir können das Package mit dem Befehl `python -m pip install -e .` bauen

Ein virtuelles Environment erstellen und das Package installieren. der -e Flag bedeutet editable, dh. Änderungen am Code werden sofort wirksam.

```
python -m pip install -e .
```

Dann einen python interpreter starten und das Package importieren

```
from testpro import prodder
prodder.say()
```

pyproject.toml Scripts

In der Datei pyproject.toml können Skripte mithilfe entsprechender Einträge (z. B. [project.scripts]) definiert werden, um sie als ausführbare Befehle zu installieren. Auf diese Weise lassen sich benutzerdefinierte Kommandos erstellen, die nach der Installation des Pakets automatisch zur Verfügung stehen.

```
[project.scripts]
prodder = "testpro.prodder:say"
```

nochmal installieren, da diese Änderungen nicht wirksam sind im edit-mode

```
python -m pip install -e .
```

und nun kann das programm im verzeichnis mit prodder aufgerufen werden

```
prodder
```

Metadata

mit der importlib können metadatan aus der pyproject.toml ausgelesen werden

```
import importlib.metadata
metadata = importlib.metadata.metadata('testpro')
print(metadata['Name'])
print(metadata['Version'])
print(metadata['Author'])
```

Projekt builden, um einen Upload auf Pypi vorzubereiten

```
pip install build
python -m build --wheels
```

Package mit twine packen, um es auf PyPi zu veröffentlichen

```
pip install twine
```

Auf test-pypi hochladen, um zu sehen, ob alles funktioniert

```
python -m twine upload --repository testpypi dist/*
```

Und dann upload auf pypi

```
twine upload dist/*
```

https://twine.readthedocs.io/en/latest/