Packages 1: Grundlagen

Proseminar „Python“

Helena Becker

Arbeitsbereich Wissenschaftliches Rechnen
Fachbereich Informatik
Fakultät für Mathematik, Informatik und Naturwissenschaften
Universität Hamburg

2022-06-14

Gliederung

Einführungsfragen

- Wozu und wie werden **Module** verwendet?
- Wie importiert man die Inhalte eines Moduls?
- Wie geht man mit einem Modul um, wenn es als **Skript** ausgeführt werden soll?
- Was ist die **Funktion** eines Packages?
- Wie kann man auf die Inhalte eines Packages zugreifen?
- Durch welches Programm lassen sich die Mehrheit der Packages installieren?

Was sind Python-Module?

- Dateien mit Endung auf .py
- Beinhalten Definitionen von Klassen, Funktionen und Variablen
  - Eigener Namensraum für Variablen
- Nach Konvention klein geschrieben
- Import ins main Modul oder in andere Module

```
1 import namemodul
```

Listing 1: Import-Syntax eines Moduls

[1, 2]

Exkurs: Funktion einer Symboltabelle

- Von Übersetzerprogramm verwendete Datenstruktur, welche jedem Symbol im Programmtext Daten zuordnet
  - Datentyp, Position des Auftretens (Verknüpfung mit Schlüssel)
- Häufig als Hash-Tabelle implementiert
- Python Funktionen: dir(), globals(), locals()

[4, 5, 6]

Einschränkungen Import Statement

- Inhalte des Moduls nicht direkt zugänglich
  - Eigene Symboltabelle
  - Punktnotation erforderlich
  - Einmalige Ausführung während Import

```
1 import namemodul
2 namemeodul.beispielfunktion('string')
3 namemodul.beispielint
```

Listing 2: Objekte des Moduls benötigen diesen als Präfix

```
1 import namemodul1, namemodul2
```

Listing 3: Import mehrerer Module

[1, 2, 3]

Alternative Varianten des Import Statements

```
1 import namemodul as namemodulneu
```

Listing 4: import ... as ...

```
1 from namemodul import beispielfunktion
```

Listing 5: from ... import ...

```
1 from namemodul import *
```

Listing 6: from ... import * -> nicht empfohlen

```
1 from namemodul import beispielfunktion as
    ↪ importfunktion
```

Listing 7: from ... import ... as ...

[1, 2, 3]

Suchpfad

- Vorgang nach der Ausführung des import Statements
  - Suche nach numpy.py in einer Liste von Verzeichnissen

```
1 import sys
2 import numpy
3 sys.path
```

Listing 8: Zugriff durch sys.path

- Beispiel-Ausgabe

['', '/content', '/env/python',
'/usr/lib/python37.zip',
'/usr/lib/python3.7',
'/usr/lib/python3.7/lib-dynload',
'/usr/local/lib/python3.7/dist-packages',
'/usr/lib/python3/dist-packages',
'/usr/local/lib/python3.7/dist-packages/IPython/extensions',
'/root/.ipython']

[1, 2]

Bestimmung des Speicherorts

- Speicherort muss in sys.path vorhanden sein

```
1 import camelcase
2 camelcase.__file__
```

Listing 9: Anwendung des Attributes __file__

- Beispiel-Ausgabe

  /usr/local/lib/python3.7/dist-packages/camelcase/__init__.py

[1, 2]

Anwendung von dir()

- Rückgabe einer Liste der Attribute und Methoden in der lokalen Symboltabelle

```
1 import camelcase
2 dir(camelcase)
```

Listing 10: Beispiel camelcase

- Beispiel-Ausgabe

```
'['CamelCase', '__builtins__', '__cached__',
'__doc__', '__file__', '__loader__',
'__name__', '__package__', '__path__',
'__spec__', 'main']
```

[1, 5]

Erneute Laden eines Moduls

- Einmaliger Import eines Moduls
- Interpreter muss neu gestartet oder Modul neu geladen werden nach einer Überarbeitung

```
1 import imp
2 import namemodul
3
4 imp.reload(namemodul)
```

Listing 11: Modul imp erforderlich

[2, 3]

Ausführen eines Moduls als Skript

- Unterscheidung, ob als Modul geladen oder als eigenständiges Skript ausgeführt wird und eine Ausgabe benötigt
- beispiel.py

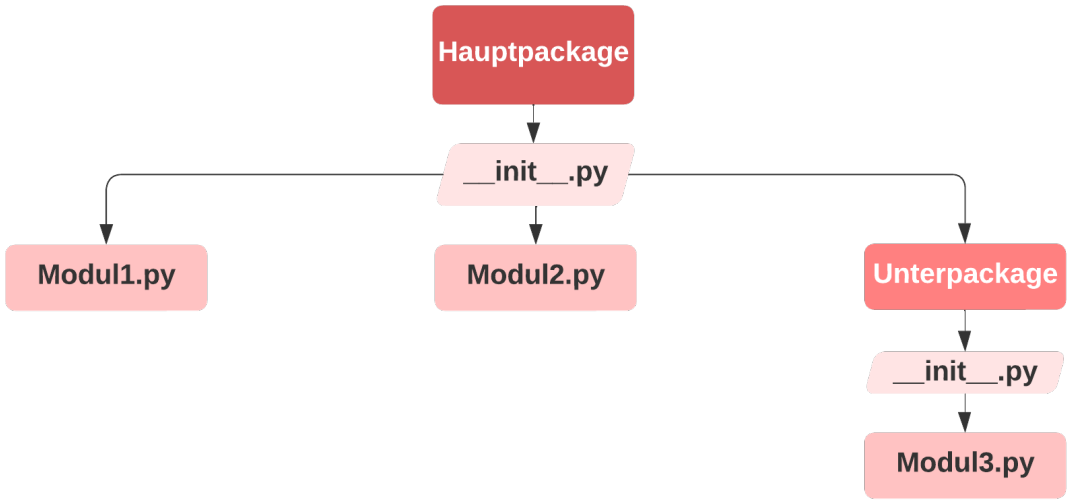
```
1 s = 'Irgendein Satz'
2
3 if(__name__ == '__main__'):
4     print(s)
```

Listing 12: Überprüfung durch ein If Statement

[1, 2]

Funktion eines Packages in Python

- Hierarchische Verzeichnisstruktur von Modulen sowie möglichen Unterpaketen
  - Zudem Dokumentation, Tests, Top-Level-Skripte, ggf. benötigte Dateien, Dateien für Installationsroutine
- Vermeidung von Kollisionen zwischen Modulnamen
- Gruppierung, Organisation und vereinfachte Nutzung von Modulen



[1, 2, 3, 7, 8]

Empfehlung einer Package-Struktur

```
Beispiel_Package/
|
|- docs
|  |- conf.py
|  |- index.rst
|  |- installation.rst
|  |- modules.rst
|  |- quickstart.rst
|  |- reference.rst
|
|- beispiel_package/
|  |- main.py
|  |- ...
|
|- tests/
|  |- test_main.py
|  |- ...
|
|- CHANGES.rst
|- LICENSE
|- README.rst
|- requirements.txt
|- setup.py
|- TODO.rst
```

[3]

Bedeutung von __init__.py

- Verzeichnisse, die __init__.py enthalten, werden aufgrund dessen als Packages behandelt
  - Nach Import des Packages oder eines Moduls des Packages, wird diese aufgerufen und dient:
    - zur Packageinitialisierung
    - zum automatischen Import von Modulen
    - und/oder zur Setzung der Variable __all__
  - Kann auch leer sein
- Seit Python 3.3 nicht mehr erforderlich

[1, 2]

Verwendung

- Zugriff auf Module (oder auch Unterpakete) durch Import-Anweisung
  - Punktnotation erforderlich

```
1 import namepaket.namemodul
2 from namepaket import namemodul
```

Listing 13: Import-Syntax eines Moduls aus einem Package

```
1 import namepaket.nameunterpaket.namemodul1
2
3 #nameunterpaket als Verzeichnis
4 from . import namemodul2
5
6 #namepaket als Verzeichnis
7 from .. import namemodul3
```

Listing 14: Relative Pfadangabe

[1, 2, 3]

Alle Module eines Packages importieren

```
1 from namepaket import *
```

Listing 15: Allgemeine Struktur

- Nur ___init___.py eines Packages wird ausgeführt
  - Kann eine Liste von Modulen namens **___all___** definieren

```
1 __all__ = ['modul1', 'modul2', 'modul3']
```

Listing 16: namepaket/___init___.py

[1, 2, 8]

Anwendung von __path__

- Unterstützung des Attributs __path__
  - Liste enthält Name des Verzeichnisses, dass die __init__.py des Packages enthält, bevor diese ausfgeführt wird
  - Beispiel: ['/usr/local/lib/python3.7/dist-packages/camelcase']
- Verzeichnis des Packages ist modifizierbar
  - Auswirkung auf die zukünftige Suche nach Modulen und Unterpackages des Packages
  - Satz von Modulen kann so erweitert werden

[1, 2, 9]

Packageinstallation

- Verwendung von PIP (Paketverwaltungsprogramm)
- Packages werden global für die gesamte Python Umgebung verwendet
  - Packages sind auf der Seite pypi.org zufinden

```
1 pip install namepackage
2 pip3 install namepackage
```

Listing 17: Installations-Syntax eines Packages

```
1 pip uninstall namepackage
2 pip3 uninstall namepackage
```

Listing 18: Deinstallations-Syntax eines Packages

[3, 10]

Literatur

[1] Python Documentation Modules and Packages
https://docs.python.org/3/tutorial/modules.html
letzter Zugriff am 22. Mai 2022

[2] Python Modules and Packages von Real Python
https://realpython.com/python-modules-packages/
letzter Zugriff am 22. Mai 2022

[3] Module und Pakete von Grund-Wissen
https://www.grund-wissen.de/informatik/python/module-und-pakete.html
letzter Zugriff am 22. Mai 2022

[4] Bedeutung Symboltabellen in Python
https://stackoverflow.com/questions/9085450/symbol-table-in-python letzter Zugriff am 22. Mai 2022

Literatur

[5] Funktion von dir()
https://www.geeksforgeeks.org/python-dir-function/
letzter Zugriff am 22. Mai 2022

[6] Symboltabelle Wikipedia
https://de.wikipedia.org/wiki/Symboltabelle letzter Zugriff am 23. Mai 2022

[7] Python Packages von Python Geeks
https://pythongeeks.org/python-packages/ letzter Zugriff am 25. Mai 2022

[8] Python-Packaging-Tutorial
https://python-packaging-tutorial.readthedocs.io/en/latest/setup_py.html letzter Zugriff am 25. Mai 2022

Literatur

[9] Verwendung von __path__ https://stackoverflow.com/questions/2699287/what-is-path-useful-for letzter Zugriff am 25. Mai 2022

[10] Nutzen von PIP https://hellocoding.de/blog/coding-language/python/pip letzter Zugriff am 25. Mai 2022

Zusammenfassung

- **Module** organisieren den Quellcode eines Softwareprojektes und beinhalten Klassen, Funktionen und Variablen.
- Falls ein Modul als **Skript** ausgeführt werden soll, so wendet man ein If-Statement an und ermöglicht so eine Ausgabe.
- Ein **Package** fasst Module sowie Unterpackages zusammen und ermöglicht so eine vereinfachte Nutzung von Modulen.
- Die Inhalte von Modulen sowie von Packages sind durch Punktnotation zugänglich.
- Durch **PIP** lassen sich die Mehrheit der Packages installieren.