Skriptsprachen: Python
Datenstrukturen

Jan Krüger, Alexander Sczyrba

Technische Fakultät
Universität Bielefeld

12. Februar 2019

Python Packaging Authority

- Erweitert die Standard Bibliothek von Python
- Python3 und Python2

PIP

- Vereinfacht die Installation von PyPa Paketen
- Support für die jeweils letzte Version ab Python 2.6
- Unterstützt (und installiert) virtualenv
  ```
  pip install virtualenv
  ```
- Unterstützt die Installation von Paketen ins Benutzerverzeichnis (keine Rootrechte notwendig)
  ```
  pip install --user package
  ```
  installiert nach `~/.local`

PIP (in der Praxis)

- Pfad erweitern:
  `export PATH=PATH:~/.local/bin`
  Achtung! Nicht persistent!
- **besser:** Eintrag in die Shell Konfigurationsdatei

VirtualEnv (via PIP)

. . . ist ein Möglichkeit isolierte Python Environments zu erstellen.

- kompatibel mit Python 3 und 2
- löst das package-dependency-hell Problem
- strikt getrennte Umgebung

VirtualEnv kann einfach mit pip installiert werden:

```
pip install --user virtualenv
```

VirtualEnv benutzen

```
mkdir <ENV>
virtual <ENV>
```

- erstellt eine neue virtuelle Umgebung
- `bin` und `include` enthalten Executables, Bibliotheken, Pakete, ...
- bin enthält ausführbare Dateien u.a. python(!), pip und activate

VirtualEnv aktivieren

```
source <ENV>/bin/activate
```

- aktiviert die neue virtuelle Umgebung in dem aktuellen Terminal
- `PATH` und `PYTHONPATH` werden angepasst
- verändert den Prompt

VirtualEnv deaktivieren

```
deactivate
```

- deaktiviert die virtuelle Umgebung
- *Resettet* die Umgebungsvariablen und den Prompt

VirtualEnv (Python 3[.6])

```
python3 -m venv <ENV>
```

- Bestandteil von Python 3.6 (seit 3.3 optionales Paket)
- schlanker
- weniger strikt als das Paket virtualenv
  - erlaubt das *Mischen* von globalen Paketen und der virtuellen Umgebung
  - `bin` enthält Links, keine Kopien

Übung

Erstellt eine virtuelle Pythonumgebung und installiert das Paket **mypy** in dieser.

- Wohin wurde das Paket in der VirtualEnv installiert ?
- Was macht dieses Paket ?
- Was ist die Motivation für ein solches Paket ?

Dynamischer vs. Statischer Typcheck

Dynamisch

```
a = "eins"
print(type(a))
<class 'str'>
```

Statisch

```
String a = "eins"
System.out.println(a.getClass().getName())
java.lang.String
```

Dynamischer vs. Statischer Typcheck (cont.)

Welche Zeile führt zu einem Fehler ?

```
b = a + 1
c = a + foo(2)
```

Falls wie foo wie folgt definiert ist:

```
def foo(p):
    return p/2
```

Type Hints (Python 3.6)

Variablen

- `a:str = "eins"`
- `b:int = 1`
- `c:bool = True`

und Funktionen

- ```
  def say(text:str) -> None:
      print(text)
  ```
- ```
  def foo(p:float) -> float:
      return p/2
  ```

Übung

- Testet die Beispiele von den vorherigen Folien in `ipython`. Benutzt TypeHints. Macht absichtlich Fehler ! Was fällt auf ?
- Erzeugt zwei Python Skripte (mit korrekter und fehlerhafter Typisierung). Benutzt `mypy` auf diesen Skripten ...