Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Ausarbeitung

Grundlegendes Python Setup

venv, pip, argparse, YAML config files

Jonas Lefert

Proseminar: Softwareentwicklung in der Wissenschaft

Studiengang: Software-System-Entwicklung
Matrikelnummer: 7199701

Betreuer: Jannek Squar

Inhaltsverzeichnis

1 Einführung - Python

Python ist eine höhere Programmiersprache, die von dem niederländischen Softwareentwickler Guido van Rossum entwickelt wurde.[1] Heutzutage hat die Python Software Foundation die Rechte an der Programmiersprache und entwickelt sie stetig weiter (Letzter Release zum Zeitpunkt der Recherche Python 3.8.5 vom 20. Juli 2020 [2]). Bei der Entwicklung wurden sich die Ziele gesetzt, den Code einfach lesbar und übersichtlich zu halten. Hieraus entwickelte sich ein anderer Code-Stil als bei anderen bekannten Programmiersprachen[1]. Charakteristisch für Python-Code ist zum Beispiel:

- Fehlen von Semikolons
- Fehlen von Rückgabetypen
- Dynamische Typisierung
- Strukturierung per Einrückung, im Gegensatz zu geschweiften Klammern

Zu sehen in Abbildung 1.1.
Zudem unterstützt Python mehrere Programmierparadigmen. Unter anderem:

- Objektorientiert
- Prozedural
- Funktional

[3]

```
# Python 3: Fibonacci series up to n
>>> def fib(n):
>>>     a, b = 0, 1
>>>     while a < n:
>>>         print(a, end=' ')
>>>         a, b = b, a+b
>>>     print()
>>> fib(1000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
```

Abbildung 1.1: Python-Code mit charakteristischem Stil [4]

[1] z.B. Java, C#, etc.

Anders als bei anderen Industriestandardsprachen wird Python üblicherweise interpretiert und nicht kompiliert. Python ist stark in der Industrie vertreten und Kenntnisse sind häufig gesucht. Als Beispiel: Python ist grundsätzlich unter den Top-5 der meistverwendeten Webentwickler-Sprachen in deutschen Firmen [5]. Hierbei wird meistens nicht nur mit den Standardfunktionen von Python gearbeitet, sondern mit Packages, die oftmals auf bestimmte Bereiche zugeschnitten sind.

2 pip - Package Installer

2.1 Installation

Pip wird dafür verwendet Packages für Python zu installieren und damit zu erweitern. Ab den Python-Versionen 2.7.9 und 3.4 wird pip standardmäßig mitinstalliert. Benutzt man andere Python-Versionen muss pip allerdings manuell dazu installiert werden. Dazu muss die Datei get-pip.py von `https://bootstrap.pypa.io/get-pip.py` heruntergeladen werden und in der Kommandozeile des jeweiligen Betriebssystems mit dem Aufruf

```
1 python get-pip.py
```

ausgeführt werden. Falls dies nicht funktioniert, muss Python zu den Systemvariablen hinzugefügt werden.

Pip wird stetig weiterentwickelt und muss somit auch regelmäßig geupdatet werden. Der Vorgang unterscheidet sich leicht zwischen Betriebssystemen. Für Mac und Linux muss der Befehl

```
1 pip install -U pip
```

und auf Windows

```
1 python -m pip install -U pip
```

ausgeführt werden. Hierbei ist Groß- und Kleinschreibung zu beachten. [6]

2.2 Benutzung

Der Großteil der Packages wird im Python Package Index (`https://pypi.org`) gesammelt. Diese können mit Hilfe von pip mit dem Befehl

```
1 pip install <Name des Packages>
```

installiert werden. Hierbei können auch andere Versionen oder Mindestversionen spezifiziert werden

```
1 pip install <Package>==1.0.4      # Installation der Version
    ↪ 1.0.4
2 pip install '<Package>>=1.0.4'    # mindestens Version
    ↪ 1.0.4, aktuellere sind auch möglich
```

Sollen in einem Projekt mit mehreren Personen oder verschiedenen Systemen die gleichen Versionen benutzt werden, kann dies in eine Requirements-Datei geschrieben werden, die mit dem Befehl

```
1 pip freeze > <Name der Datei>.txt
```

erstellt wird.

Um alle Packages zu installieren, die in einer Requirements-Datei gespeichert sind, muss der Befehl

```
1 pip install -r <Name der Datei>.txt
```

benutzt werden.

Ähnlich wie eine Requirements-Datei, kann auch eine Constraints-Datei verwendet werden. Der Unterschied zwischen diesen ist, dass das Verwenden einer Constraints-Datei nicht immer alles installiert. Bevor die Packages installiert werden, wird überprüft ob das Projekt die Packages in der Constraints-Datei benötigt und es werden nur jene installiert die das Projekt wirklich benötigt oder von denen eine andere Version vorliegt. Der Aufruf um mit einer Constraints-Datei Packages zu installieren ist:

```
1 pip install -c <Name der Constraints-Datei>.txt
```

Um sich anzeigen zu lassen, welche Packages installiert sind, wird der Befehl

```
1 pip list
```

benutzt.

Code-Beispiele aus dem pip User-Guide (Siehe [7])

3 Virtual Environments - venv

Virtual Environments sind isolierte, geschlossene Installationen von Python, in der die installierte Pythonversion und die Abhängigkeiten und Versionen der Packages unabhängig von der Hauptinstallation von Python sind. Virtual Environments werden verwendet, weil viele Packages nicht mit der installierten Pythonversion kompatibel sind oder bestimmte möglicherweise veraltete Versionen in einem Projekt verwendet werden. Somit kann man einzelne Unterprogramme eines Projekts auf verschiedenen Versionen von Python und unterschiedlichen Packages ausführen. Dies löst zudem auch das Problem, dass einige Packages nicht miteinander kompatibel sind, aber trotzdem in einem Projekt benutzt werden sollen. Um Virtual Environments zu erstellen, gibt es mehrere Packages, die mit Hilfe von pip installiert werden können. Ab der Pythonversion 3.3 ist venv auch in der Standardbibliothek integriert. [8]. Ein bekanntes Package, was diese Aufgabe erfüllt, ist virtualenv. Zum Installieren mit pip wird der Befehl

```
1 pip install virtualenv
```

ausgeführt. Um nun ein Virtual Environment mit diesem Package zu installieren muss man zuerst mit der Kommandozeile in den Ordner in dem das Environment erstellt werden soll. Daraufhin muss der Befehl

```
1 virtualenv <Name des Environments>
```

ausgeführt werden. Dies erstellt einen Ordner mit einer Kopie der installierten Pythonversion mit pip ohne weitere installierte Packages in dem geöffneten Ordner. Möchte man nun mit dem Virtual Environment arbeiten, muss dieses erst aktiviert werden. Dazu muss der Ordner des Virtual Environments geöffnet werden und die Datei activate.bat über die Konsole ausgeführt werden. Nun können für dieses Environment Packages installiert werden, die isoliert von der Haupt-Pythonversion sind. Um das Environment zu verlassen muss dieses lediglich mit

```
1 deactivate
```

deaktiviert werden. Wie anfangs beschrieben können auch unterschiedliche Pythonversionen verwendet werden. Dazu muss die gewünschte Pythonversion installiert werden. Daraufhin wird bei der Erstellung ein optionaler Parameter hinzugefügt

```
1 virtualenv -p <Verzeichnis mit der gewünschten
  ↪ Pythonversion> <Name des Environments>
```

[9]

4 argparse

In der Standardbibliothek von Python sind verschiedene Packages enthalten, die das Arbeiten mit Python angenehmer gestalten. Eines davon ist argparse. Es wird dazu verwendet Programme mit Kommandozeileninterfaces zu schreiben. Hierbei wird im Programmcode festgelegt wie Argumente verarbeitet und zurückgeliefert werden sollen. Dies geschieht mit einem Argument Parser. Diesem können Argumente hinzugefügt, die entweder optional oder verpflichtend sind. Den Argumenten können bestimmte Eigenschaften gegeben werden, die das Verhalten und die Art dieser verändern. Als Beispiel:

```
1  import argparse
2
3  parser = argparse.ArgumentParser(description='Process some
       ↪ integers.')
4  parser.add_argument('integers', metavar='N', type=int,
       ↪ nargs='+',
5                      help='an integer for the accumulator')
6  parser.add_argument('--sum', dest='accumulate',
       ↪ action='store_const',
7                      const=sum, default=max,
8                      help='sum the integers (default: find
                          ↪ the max)')
9
10 args = parser.parse_args()
11 print(args.accumulate(args.integers))
```

[10]
In Zeile 1 wird das Package in das Pythonprogramm importiert, damit es benutzt werden kann. In Zeile 3 wird ein neuer Argument Parser erstellt, der bei Hinzufügen des optionalen help-Parameters den String ausgibt, der hinter description spezifiziert wurde. In Zeile 4 wird dem Parser ein neues Argument mit dem Namen integers hinzugefügt. Dieses ist verpflichtend beim Aufruf anzugeben, da die Zeichen fehlen, die Optionalität angeben (standardmäßig '-'). Daraufhin wird festgelegt, dass integers in der Kommandozeile als 'N' angezeigt wird und eine Ganzzahl ist. Als type können die Standardtypen[1] gewählt werden. Wird type nicht angegeben ist der Standardtyp eines Arguments String. Folgend wird mit dem '+' festgelegt, dass mindestens ein Argument gegeben sein muss, aber auch mehrere möglich sind. Auch für nargs sind verschiedene Optionen möglich (Siehe

[1] int, boolean, float, etc.

[10]). Am Ende wird der Text gesetzt, der die Variable beschreibt, wenn die Hilfe zum Programm angezeigt wird.

In Zeile 6 wird ein zweites optionales Argument zum Parser hinzugefügt. Dies ist ersichtlich, da der Name das '-'-Zeichen enthält. Die Optionalitätszeichen können auch mit dem Hinzufügen des Parameters prefix_chars beim Erstellen des Argument Parsers selbstständig verändert werden. Falls ein optionales Argument hinzugefügt wird, muss zusätzlich festgelegt werden, wie der interne Name der Funktion ist mit der das Argument ausgeführt werden kann. Dies übernimmt der dest-Parameter. Dann wird spezifiziert was passieren soll, falls der optionale Parameter beim Aufruf benutzt wurde. In diesem Fall wird der benutzte Wert auf den im const-Parameter spezifierten Wert gesetzt. Auch hier existieren wieder mehrere Möglichkeiten. Default gibt an, welcher Wert benutzt wird, wenn der optionale Parameter nicht übergeben wird. Zum Schluss wird wieder der Text festgelegt, der beim Hilfeaufruf angezeigt wird.

Daraufhin werden die Argumente die beim Aufruf vom Nutzer übergeben werden an eine Variable übergeben. Mit dieser Variable können dann Operationen mit den Argumenten ausgeführt werden. In diesem Fall wird das Maximum der übergebenen Zahlen zurückgegeben, wenn der optionale Parameter nicht mit übergeben wurde und die Summe aus integers berechnet wenn der optionale vorhanden ist.

Mit dem argparse Package können somit Pythonprogramme mit User-Input in der Kommandozeile programmiert werden und dem Nutzer kann eine gute Dokumentation gewährleistet werden, im Gegensatz zu anderen Kommandozeilen-Programmen.

5 YAML - Konfigurationsdateien

Für viele Projekte ist es nützlich eine Art der Serialisierung zu haben um Daten zu speichern. Einerseits um ähnliche Ausführungskonfigurationen zu gewährleisten oder Daten, die in vielen Dateien verwendet werden schnell ändern zu können. Andererseits um Daten zwischen mehreren Projektteilnehmern schnell zu verteilen. Hilfreich hierfür ist eine Markup-Language, die Daten strukturiert und so gut wie möglich menschenlesbar abspeichern. Eine derartige Markup-Sprache ist YAML. YAML hat eine stark vereinfachte Syntax im Gegensatz zu anderen bekannten Markup-Sprachen wie XML oder HTML, was YAML einfach benutzbar macht. Zudem ist sie durch die Einfachheit gut in verschiedene Sprachtypen übersetzbar.

Dateien sollten grundsätzlich auf .yaml enden und in diesen werden Tabulatoren erlaubt, sondern lediglich Leerzeichen.

Eine YAML-Datei ist in Dokumente aufgeteilt, die mit drei Bindestrichen ('—') beginnen. In diesen können theoretisch unendlich Daten gespeichert werden. Zudem können in einer .yaml-Datei mehrere Dokumente gespeichert werden. Diese müssen jeweils mit '—' begonnen werden.

YAML unterstützt drei Dateitypen:

- Skalare
- Sequenzen
- Mappings

[11]

Skalare umfassen die grundlegenden Datentypen[1]. Hierbei können Integer auch als Hex- (0xXXX) bzw.[2]beziehungsweise Oktalzahlen (0XXX) repräsentiert werden. Strings müssen zudem nicht zwingend mit Anführungszeichen spezifiziert werden. Skalare beginnen mit dem Namen des Skalars gefolgt von einem ':' und, getrennt mit einem Leerzeichen, dem Wert des Skalars.

Sequenzen sind einfache Listen, in denen jeder Dateityp gespeichert werden kann. Das heißt verschachtelte Listen sind möglich. Listenelemente beginnen mit einem '-' und dem Wert des Listenelements dahinter. Elemente einer Liste werden durch Zeilenumbrüche getrennt. Möchte man eine Liste verschachteln muss man lediglich ein Leerzeichen vor das nächste Listenelement setzen.

Mappings sind Hashtabellen bzw.[2] Dictionaries. Das heißt ein Wert wird einem Schlüssel zugeordnet, wodurch das Suchen und Finden eines Elements schneller wird. Einem Key kann auch eine Sequenz zugeordnet werden oder auch ein weiteres Mapping.

[1]int, float, String, char, etc.

Beispiel einer YAML-Datei:

```
---                         # Beginn eines Dokuments
 int: 12345
 hex: 0x1fd2
 oct: 02743
 string1: String
 string2: "String"          # identisch mit string1
 float: 3.14159
 boolean: true
 -                          # verschachtelte Liste
  - 1
  - "String"
  - true
 tier: fische
 fische:
  - Barsch
  - Wels
```

Um YAML-Dateien in Python einzulesen und auszugeben, gibt es das Package PyYAML[12]. PyYAML lädt die Daten aus einer .yaml Datei aus einem Datenstream und schreibt diesen dann in ein Dictionary um, welches dann in Python benutzt werden kann. Beispiel:

```
import yaml

stream = open("<Name>.yaml", 'r')

dictionary = yaml.load(stream, Loader = yaml.FullLoader)
```

[2]beziehungsweise

Literaturverzeichnis

[1] Python (Programmiersprache). Accessed: 2020-08-17. [Online]. Available: https://de.wikipedia.org/wiki/Python_(Programmiersprache)

[2] Python Downloads. Accessed: 2020-08-17. [Online]. Available: https://www.python.org/downloads/

[3] What is Python? Accessed: 2020-08-17. [Online]. Available: https://docs.python.org/3/faq/general.html#what-is-python

[4] Python.org. Accessed: 2020-08-17. [Online]. Available: https://www.python.org

[5] T. W. (2017-11-29), "Das sind die gefragtesten Programmiersprachen in Unternehmen," *t3n - digital pioneers*, Accessed: 2020-08-17. [Online]. Available: https://t3n.de/news/programmierer-gesucht-developer-studie-programmiersprache-881255/

[6] pip Installation. Accessed: 2020-08-17. [Online]. Available: https://pip.pypa.io/en/stable/installing/

[7] pip - User Guide. Accessed: 2020-08-17. [Online]. Available: https://pip.pypa.io/en/stable/user_guide/

[8] Virtual Environments and Packages. Accessed: 2020-08-18. [Online]. Available: https://docs.python.org/3/tutorial/venv.html

[9] C. S. (2015-04-13). Python Tutorial: virtualenv and why you should use virtual environments. Accessed: 2020-08-18. [Online]. Available: https://www.youtube.com/watch?v=N5vscPTWKOk

[10] argparse — Parser for command-line options, arguments and sub-commands. Accessed: 2020-08-18. [Online]. Available: https://docs.python.org/3/library/argparse.html#module-argparse

[11] YAML syntax. Accessed: 2020-08-18. [Online]. Available: https://learn.getgrav.org/16/advanced/yaml

[12] PyYAML. Accessed: 2020-08-18. [Online]. Available: https://pypi.org/project/PyYAML/