Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Ausarbeitung

Virtual Environments

vorgelegt von

Youssef Al Shriteh

Fakultät für Mathematik, Informatik und Naturwissenschaften
Fachbereich Informatik
Proseminar „Python im Hochleistungsrechnen“

Studiengang: Informatik
Matrikelnummer: 6973934

Betreuer: Dr. Michael Kuhn

Hamburg, 23.08.2019

Contents

1 Einleitung

Als Python-Programmierer waren Sie sicherlich in einer Situation, in der Sie möchten, dass Ihre Software an Python 3.5- und 2.7-Versionen funktioniert, oder Sie möchten einen Code mit Ihrer installierten Bibliothek, aber mit einer niedrigeren oder höheren Version erstellen. Werden Sie in diesem Fall die Bibliothek löschen und sie mit der gewünschten Version nochmal installieren?
Was ist mit dem ersten Fall, den einige Programmierer für erschreckend halten, wo man sich fragen muss, wie wir zwei Versionen von Python auf demselben Betriebssystem installieren werden, und wie wir damit umgehen werden?!
Mit Virualenv kann man einfach mehr als eine Version von Python auf demselben Betriebssystem installieren, mit der Möglichkeit, die Version auszuwählen, an der man arbeiten möchte.
An dieser Stelle ist es gut zu erwähnen, dass es drei Arten von Virtuellen Umgebungen gibt.

1. Virtuelle Maschinen, die einen kompletten Rechner mit Betriebssystem bereitstellen.
2. Container, die nur die nötigste Software enthalten.
3. Virtuelle Umgebungen, die nur benötigten Bibliotheken für Programmiersprachen bereitstellen.

Wir beschäftigen uns hier mit der dritten Art.

2 Virtual Environment

Python bietet uns ein großartiges Virtualenv-Tool, mit dem wir isolierte, separate Python-Umgebungen erstellen können, die alle grundlegenden Dateien und Bibliotheken enthalten, und zum Erstellen von Python-Software erforderlich sind. Das bietet uns natürlich hohe Dynamik beim Erstellen von Software.

1. **Verschiedene Bibliothek-Versionen auf derselben Python-Version verwenden:**
   Wir geben ein Beispiel für eine Situation, in der die Bedeutung der Verwendung der virtuellen Umgebung offensichtlich ist.
   Angenommen, wir müssen numpy-Bibliothek verwenden, um ein System zu bauen, das mit Bibliotheksversion 1.16.4 und Version 1.16.2 kompatibel sein muss.
   Man kann die Bibliotheksversionen nicht in derselben Python-Version installieren, da die Art des Erstellens und Verwendens externer Bibliotheken in Python eine einzelne Version einer Bibliothek in derselben Version erfordert.
   In diesem Fall verwenden wir das Virtualenv-Tool, um Python-Umgebungen zu erstellen und die beiden gewünschten Versionen aus der numpy-Bibliothek in jeder Version in einer Umgebung zu installieren. Wir nehmen an, dass die Version 1.16.4 auf Environment A und Version 1.16.2 auf Environment B installiert ist. Um den Code für das geplante System auf Bibliotheksversion 1.16.4 auszuführen, aktivieren wir Environment A. Auf Bibliotheksversion 1.16.4 aktivieren wir Environment A. Nachdem wir die Arbeit an Environment A abgeschlossen haben aktivieren wir Environment B, um das System mit Version 1.16.2 der Bibliothek zu betreiben.

   Die Python-Version, die Beim Erstellen der Umgebung verwendet wird, ist 3.7.3 und das Betriebssystem ist Windows 10, und die Befehle werden im CMD ausgeführt.

- **Installation von virtualenv:** Zuerst muss man virtualenv installieren, um virtuelle Umgebungen erstellen und verwalten zu können. Das können wir durch den folgenden Befehl machen.

```
1 pip install virtualenv
```

Listing 2.1: Installation von virtualev

Um sicherzustellen, dass virtualenv erfolgreich installiert wurde:

```
1 virtualenv --version
```

Listing 2.2: Version von virtuelenv

Das soll die installierte Version von virtualenv zurückgeben.

- **Erstellen ener Umgebung:** Um die erste Umgebung A zu erstellen, die wir envA nennen werden, geben wir den folgenden Befehl ein:

```
1 virtualenv envA
```

Listing 2.3: Virtuelle Umgebung envA erstellen.

Der vorherige Befehl erstellt einen neuen Ordner namens envA und dann erstellt Umgebungsdateien. Der Ordner wird eine neue und Miniatur-Python-Umgebung der Version 3.7.3 enthalten. Es wird Standard Python-Bibliotheken und pip enthalten und keine externen Bibliotheken.
Wenn die Umgebung aktiviert ist, ist die Verwendung von pip spezifisch für diese Umgebung, und ihre Verwendung hat keinen Einfluss auf die Hauptversion von Python auf dem Gerät. Wir werden dasselbe für Umgebung envB tun:

```
1 virtualenv envB
```

Listing 2.4: Virtuelle Umgebung envB erstellen.

Wir haben jetzt zwei Umgebungen, auf die wir die benötigten Bibliotheken installieren können. Aber vorher müssen wir wissen, wie wir die gewünschte Umgebung aktivieren können.

- **Eine virtuelle Umgebung aktivieren:** Um die Umgebung envA zu aktivieren, führen wir die aktive Patchdatei aus, die sich im Skript-Ordner im Umgebungsordner befindet.

```
1 envA\Scripts\activate
```

Listing 2.5: Virtuelle Umgebung envA aktivieren.

Die Befehlszeile wird durch den aktiven Umgebungsnamen in Klammern vorangestellt.

- **Installation einer Bibliothek mit bestimmter Version:** Jetzt können wir pip verwenden um numpy-Bibliothek in der Version 1.16.4. zu installieren. Man beachte, dass das, was wir während der Aktivierung einer Umgebung installieren, im Umgebungsordner gespeichert wird und nichts mit der Python-Version zu tun hat.

```
1 (envA)   pip install numpy==1.16.4
```

Listing 2.6: Version 1.16.4 von numpy auf envA installieren.

Jetzt können wir den gewünschten Code ausführen und sicherstellen, dass er mit der numpy-Bibliothek in Version 1.16.4 funktioniert.
Wir aktivieren die Umgebung envB

```
1 envB\Scripts\activate
```

Listing 2.7: Virtuelle Umgebung envB aktivieren.

Und installieren die Version 1.16.2 der Bibliothek numpy.

```
1 (envB)   pip install numpy==1.16.2
```

Listing 2.8: Version 1.16.2 von numpy auf envB installieren.

- **Eine Umgebung deaktivieren:** Wenn wir mit der Arbeit an der Umgebung fertig sind, und die Umgebung deaktivieren möchten, führen wir die Datei deactivate in Skripts aus.

```
1 (envB) envB\Scripts\deactivate
```

Listing 2.9: Eine Umgebung deaktivieren.

Oder einfach:

```
1 (envB) deactivate
```

Listing 2.10: Eine Umgebung deaktivieren.

So können wir einen einzelnen Code ausführen, der zwei unterschiedliche und separate Python-Umgebungen verwendet und zwei verschieden Versionen der numpy-Bibliothek verwendet, wodurch der Code mit mehr als einer Version der Bibliothek portabel ist.

2. **Verschiedene Python-Versionen verwenden:**
Falls wir den Python-Code auf Version 2.7 und 3.7 verwenden möchten, empfehlt sich, eine Umgebung für jede Version zu erstellen und unser Projekt mit Virtualenv zu erstellen und auszuführen. In diesem Fall müssen wir virtualenv mitteilen, woher die Umgebung erstellt werden soll. Also wir müssen den Pfad der Version bestimmen, von dem aus wir die Umgebung erstellen möchten. Um eine Umgebung aus der Python-Version 3.7.3, unter der Annahme, dass sie auf dem C-Datenträger installiert ist.

```
1 virtualenv -p c:\Python35\python.exe env37
```

Listing 2.11: Umgebung für Python3.7.3 erstellen.

Option p bestimmt den pfad für die Datei python.exe.
env37 ist der Name der Umgebung. Wir erstellen auch eine Umgebung für Version 2.7.

```
1 virtualenv -p c:\Python27\python.exe env27
```

Listing 2.12: Umgebung für Python2.7 erstellen.

Wir haben also eine Umgebung für Version 2.7 und eine für Version 3.7 und können die benötigte Umgebung aktivieren und deaktivieren.

3 PYTHONPATH

3.1 Definition

PYTHONPATH ist eine Umgebungsvariable, der den Standardsuchpfad für Moduldateien bereitstellt.
Diese Variable „path“ repräsentiert eine Liste von Pfaden, in deren Speicherorte nach Module gesucht wird.
Wenn wir ein Modul importieren (import <name>), durchsucht Python der Reihe nach die folgenden Speicherorte nach einem übereinstimmenden Namen:

1. **Aktuelles Arbeitsverzeichnis:** Der Ordner, der den Python-Code enthält, an dem wir arbeiten.
2. **PYTHONPATH**
3. **Default Installationsverzeichnis für Module:** Die selbst installierte Module besonders mit dem pip-Paketmanager.

Die Liste path befindet sich in dem Modul sys, also wenn wir auf diese Liste zugreifen wollen, müssen wir zuerst sys importieren.
So können wir die Liste der Suchpfaden path ausgeben lassen.

```
1 import sys
2 sys.path
```

Listing 3.1: Die Liste path

3.2 Eigenes Modul importieren (path erweitern)

Nun nehmen wir an, dass wir unser eigenes Modul entwickelt haben und wir es importieren möchten.
Es gibt drei Wege, um ein Modul für einen Import auffindbar zu machen.

1. **path-Datei anlegen:**
   Hier speichert man das Modul in einem der Pfade in path, sodass das Modul gefunden wird, wenn Python in dem entsprechenden Pfad sucht. Ein Nachteil hier ist die Unübersichtlichkeit, wenn man zu viel Module hat.

2. **Systemvariablen aus Python heraus editieren:**
Hier erweitert man die Liste der Suchpfade „path“ um einen Pfad, wo sich unsere Module befinden. Man hat zwar „path“ erweitert aber nicht dauerhaft und nur in der aktuellen Session. Das heißt man muss in jeder Session die Liste erneut erweitern.

   Das kann man wie folgt machen:

   a) **append:**

```
1 sys.path.append("/pfad/zu/unserem/modul/")
```

Listing 3.2: path mit append erweitern

   b) **insert:**

```
1 sys.path.insert(Index, "/pfad/zu/unserem/modul/")
```

Listing 3.3: path mit insert erweitern

   Der einzige Unterschied zwischen insert und append ist, dass bei append der neue Pfad als letzter Pfad an die Liste gehängt wird. Mit insert kann man noch die Position des neuhinzugefügten Pfades auswählen.

3. **Systemvariablen auf Betriebssystemebene editieren:**
Wie oben erklärt, aber nicht auf Python-Ebene sondern auf Betriebssystemebene, damit die Erweiterung dauerhaft gespeichert wird.

   a) **Windows:**
   In Windows geben wir dazu in der Suchleiste ‚Systemumgebungsvariablen‘ ein. Mit dem Button Umgebungsvariablen öffnet das Menü zum Editieren der Variablen. Nun suchen wir im unteren Bereich des Fensters unter Systemvariablen nach dem Eintrag PYTHONPATH. Abschließend ergänzen wir den Pfad um das Verzeichnis, in dem sich unsere zu importierenden Modulen befinden. Häufig ist es nach dem Anlegen der Variable nötig, Windows neu zu starten.

   b) **Linux:**
   In Linux können wir den Pythonpath mit dieser Anweisung editieren:

```
1 PYTHONPATH="${PYTHONPATH}:/pfad/zu/unserem/modul/"
2 export PYTHONPATH
```

Listing 3.4: Systemvariablen auf Linux editieren

   Nach dem editieren der Systemvariable sollte unser Verzeichnis nun im Output der Anweisung **sys.path** auftauchen.

4 PYPI/PIP

4.1 PYPI:

ist eine Abkürzung für Python Package Index, und ein offizielle Software-Repository für Python. Es wird auch Cheese Shop genannt.
PYPI enthält über 113.000 Python-Pakete und ist auch die Standardquelle für viele Paketmanager wie PIP.
Jeder kann seine eigene Module in PYPI veröffentlichen, damit andere sie verwenden können.

4.2 PIP

ist ein Paketmanager, mit dem man die Pakete verwalten kann. PIP ist eine Alternative zum easy_install. Über den folgenden Link findet man die Unterschiede zwischen PIP und easy_install.
`https://packaging.python.org/pip_easy_install/`
Ab Version 3.4 von Python ist PIP automatisch installiert. Mit PIP kann man jedes Python-Paket, das im PYPI-Repository gespeichert ist, wie folgt installieren.

```
1 pip install "Name"
```

Listing 4.1: Pakete mit PIP installieren

oder die schon installierte Pakete deinstallieren.

```
1 pip uninstall "Name"
```

Listing 4.2: Pakete mit PIP deinstallieren

Man kann auch eine Liste von den installierten Paketen ausgeben lassen.

```
1 pip list
```

Listing 4.3: Liste von den installierten Paketen

Außerdem, kann man nach bestimmtem Paket suchen oder Informationen über ein bestimmtes Paket fordern.

```
1 pip search "Name"
2 pip show "Name"
```

Listing 4.4: Pakete suchen/ Info fordern

5 ANACONDA

ANACONDA ist eine kostenlose Python-Distribution, die darauf abzielt, die Verwaltung von Python-Bibliotheken zu erleichtern. Sie ist vor allem wichtig, wenn die Chance besteht, dass man mehrere verschiedene Python-Versionen braucht. Anaconda hat viele wichtige Pakete mit sich und bietet zwei verschiedene Benutzerschnittstellen an.

1. Anaconda Navigater:
   Er bietet graphische Oberfläche an, und ist sehr einfach mit der Maus zu bedienen. Hier kann man mehrere Umgebungen für unterschiedliche Python-Versionen erstellen, und einfach zwischen denen wechseln. Man hat außerdem einen besseren Überblick auf die Pakete und die Umgebungen.

2. CONDA:
   ist ein leistungsstarker Paketmanager und Umgebungsmanager. Hier wird Kommandozeile verwendet.
   Hier sind die wichtigsten Befehle in conda:

   Man kann eine neue Umgebung erstellen mit der Angabe der gewünschten Python-Version.

```
1 conda create -n Name Python=2.7
```

Listing 5.1: Umgebung mit dem Namen Name und Python-Version 2.7

a) Man kann nach der aktuellen Conda-Version fragen.

```
1 conda --version
```

Listing 5.2: aktuelle conda-Version

b) Und aktualisieren

```
1 conda update conda
```

Listing 5.3: conda-Version aktualisieren

c) Mit diesem Befehl kann man eine Liste von den schon installierten Paketen ausgeben lassen.

```
1 conda list
```

Listing 5.4: Liste von installierten Paketen

d) man kann noch nach bestimmtem Paket suchen, installieren oder deinstallieren.

```
1 conda search Paket-Name
2 conda install Paket-Name
3 conda uninstall Paket-Name
```

Listing 5.5: Paket installieren

Die Befehle von conda sehen zwar wie die Befehle von pip aus, aber conda kann viel mehr als pip. Pip ist spezifisch für Python-Pakete und conda kann verwendet werden, um Pakete aus einer beliebigen Sprache zu verwalten. Außerdem behandelt conda Bibliotheksabhängigkeiten außerhalb der Python-Pakete sowie die Python-Pakete selbst, was pip nicht tut. Conda schafft auch eine virtuelle Umgebung, wie virtualenv tut.
Man kann auch conda und pip nebeneinander verwenden.

6 Literaturen

a) Greek for Greek, Python Virtual Environment | Introduction, [1]

b) Mathias Weidner, Eine virtuelle Umgebung für Python, [2]

c) Andreas Wygrabek, Der Pythonpath, [3]

d) Wikipedia, Python Package Index, [4]

e) ANACONDA DUCUMENTATION, Getting started with Anaconda, [5]

[1] https://www.geeksforgeeks.org/python-virtual-environment/

[2] http://weidner.in-bad-schmiedeberg.de/computer/software/python/python-virtual-environment/

[3] https://www.data-science-architect.de/pythonpath/

[4] https://en.wikipedia.org/wiki/Python_Package_Index

[5] http://docs.anaconda.com/anaconda/user-guide/getting-started/