FAQ
Mit Python entwickeln

Antworten auf die häufigsten Fragen

Von Johannes Merkert

Python 2 oder 3

? Python 3 ist moderner, von einigen Bibliotheken habe ich aber gehört, dass sie nur mit Python 2 laufen. Sollte man mit Version 2.7 oder 3.x entwickeln?

! Die Frage stellt sich durchaus, da Python 3 nicht vollständig abwärtskompatibel zu Python 2 ist. Da aber alle wichtigen Bibliotheken und Frameworks mittlerweile mit Python 3 laufen, können Sie im Normalfall getrost auf die neuere Version setzen. Ein weiteres Argument für den Nachfolger ist, dass einige Programme wie Blender kein Python 2 mehr unterstützen.

Über Imports aus dem `future`-Modul können Sie Programme schreiben, die mit beiden Versionen der Sprache laufen. Dann empfiehlt es sich, mit Python 3 zu entwickeln und hin und wieder mit Python 2 zu prüfen, ob der Code mit dem alten Interpreter noch korrekt läuft. Die beiden Versionen lassen sich auf allen Betriebssystemen parallel installieren.

Python unter Windows

? Windows wird ohne Python-Interpreter ausgeliefert. Welche Python-Distribution sollte man installieren?

! Normalerweise genügt das offizielle Installationspaket von python.org. Falls Sie jedoch in C implementierte Bibliotheken wie Numpy benötigen, wird die Installation unter Windows etwas komplizierter. Zwar bringt auch Python für Windows den Paketmanager `pip` mit. Der muss für die Installation von Numpy aber denselben C-Compiler anwerfen, mit dem auch Python kompiliert wurde. Microsoft bietet ein für einige Python-Versionen passendes Paket an, für andere müssten Sie aber ein komplettes VisualStudio installieren.

Python-Distributionen wie Anaconda bringen gleich das komplette Paket aus Python-Interpreter und vorkompilierten Bibliotheken wie Numpy oder Scipy mit. Die freie Python-IDE Spyder bringt in ihrem Windows-Installer auch alle Abhängigkeiten für den wissenschaftlichen Einsatz von Python unter.

Paketverwaltung wählen

? Sollte man Bibliotheken über den Paket-Manager des Systems installieren oder über pip?

! Der Python-Paketmanager `pip` installiert Pakete aus dem Python Package Index (PyPi) und nicht aus den Paketquellen des Systems. Die Paketmanager der Linux-Distributionen und Homebrew auf dem Mac erwarten aber die Versionen der Python-Pakete aus ihren eigenen Quellen, die oft etwas älter sind. Darüber hinaus berücksichtigen die Pip-Pakete keine automatischen Sicherheitsaktualisierungen.

Da Python-Programme zu vielen Linux-Systemen gehören, riskieren Sie Probleme, wenn Sie deren Abhängigkeiten an der Paketverwaltung vorbei mit `pip` aktualisieren. Hier sollten Sie immer die Version verwenden, die das Linux installiert. Werden die Abhängigkeiten nur von Ihren selbst entwickelten Programmen benutzt, entschärft sich die Problematik. Aber auch hier erspart es Probleme, wenn Sie mit `apt` & Co. statt `pip` installieren. Sollte das Betriebssystem ein von Ihnen benötigtes Paket nicht bereitstellen, kommen Sie an `pip` allerdings nicht vorbei. Dann sollten Sie aber auch ein Virtualenv installieren.

Virtuelle Umgebung

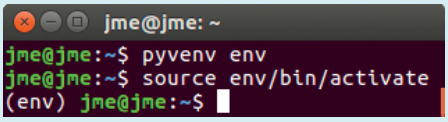
? Sollte man zum Entwickeln grundsätzlich ein Virtual Environment (Virtualenv) erstellen?

! Ein Virtualenv ist eine abgeschottete Arbeitsumgebung, in der `pip` beliebige Python-Pakete installieren kann, ohne außerhalb laufende Python-Programme zu stören. Python 2 erstellt die Umgebung mit dem Programm `virtualenv`; Python 3 verwendet `pyvenv`:

```
pyvenv env
```

Daraufhin liegt eine Kopie des Python-Interpreters samt `pip` im Verzeichnis env/. Sie können dem Verzeichnis einen beliebigen Namen geben, „env“ ist aber Standard und damit beispielsweise in GitHubs .gitignore für Python enthalten. Die geschaffene Umgebung aktiviert der Befehl

```
source env/bin/activate
```



Ob ein Virtualenv aktiviert ist, erkennen Sie an dessen Namen in Klammern am Anfang des Prompts.

Wenn Sie danach Pakete mit `pip` installieren, landen diese nur innerhalb des Virtualenv. Ihr Programm verwendet sie aber genau wie systemweit installierte Bibliotheken. Das verhindert Versionskonflikte, wenn unterschiedliche Programmierprojekte auf unterschiedliche Versionen der gleichen Bibliothen zugreifen. Sie verlassen das Virtualenv mit

```
deactivate
```

? Wie aktualisiert man alle Pakete in einem Virtualenv?

! Virtualenvs kommen beispielsweise bei Django-Projekten auf Webservern und damit auch in Produktivsystemen vor. An dieser Stelle vermisst man automatische Systemupdates. Leider bietet `pip` keine Funktion, um alle PyPi-Pakete auf dem neuesten Stand zu halten. Mit etwas Shell-Magie aktualisiert `pip` aber trotzdem alle Pakete im Virtualenv:

```
pip install --upgrade pip
pip freeze --local | grep -v '^\-e' |
                   ↳ cut -d = -f 1 |
             ↳ xargs -n1 pip install -U
```

Da die zwei Befehle alle Python-Pakete auf die aktuelle Version aktualisieren, eignen sie sich auch, um bei längeren Programmierprojekten mit der Zeit zu gehen und immer mit aktuellen Versionen zu testen. (*jme@ct.de*) c't