2. Verwendung des Python-Interpreters

2.1. Aufrufen des Interpreters

Sofern der Python-Interpreter auf einem Rechner installiert ist, findet man ihn normalerweise unter `/usr/local/bin/python/python3.3`. Wenn man `/usr/local/bin` in den Suchpfad der Unix-Shell setzt, kann man den Interpreter aufrufen durch [1]:

```
python3.3
```

Die Auswahl des Installationspfades für den Interpreter ist eine Installationsoption, so dass auch eine Installation an anderer Stelle möglich ist. Das ist mit dem örtlichen Python-Guru oder dem Systemadministrator zu klären. (Eine populäre Alternative ist etwa `/usr/local/python`)

Auf Windows-Rechnern befindet sich die Pythoninstallation meist unter `C:\Python33`, auch wenn man das während des Installationsvorgangs ändern kann. Um dieses Verzeichnis zum Suchpfad hinzuzufügen, kann man folgendes Kommando in die DOS-Eingabeaufforderung eingeben:

```
set path=%path%;C:\python33
```

Durch Eingabe eines End-Of-File-Zeichens (EOF; `Strg-D` unter Unix, `Strg-Z` unter Windows) in der Eingabeaufforderung des Interpreters wird der Interpreter mit dem Rückgabewert Null beendet. Falls er das nicht tut, kann man den Interpreter durch folgende Befehlszeile beenden: `quit()`.

Die Möglichkeiten des Interpreters hinsichtlich des Editierens der Eingabe sind ziemlich beschränkt, lassen sich aber durch Einsatz der GNU-readline-Bibliothek erweitern. Ob diese erweiterten Möglichkeiten verfügbar sind, lässt sich überprüfen, indem man ein `Strg-P` in die Eingabeaufforderung tippt. Wenn es piepst, ist die “readline”-Unterstützung vorhanden. In diesem Fall findet man im Anhang Interaktive Eingabe-Bearbeitung und Ersetzung des Verlaufs eine Einführung zu den einzelnen Tasten. Falls kein Piepton zu hören ist oder `^P` erscheint, ist keine “readline”-Unterstützung vorhanden und die einzige Möglichkeit zum Editieren ist die Verwendung der Rücktaste (Backspace), um Zeichen in der aktuellen Eingabezeile zu entfernen.

Grundsätzlich ist der Interpreter ähnlich zu bedienen wie eine Unix-Shell: Wird er mit einem tty-Gerät als Standardeingabe aufgerufen, liest und führt er interaktiv Befehle aus. Wird er mit einem Dateinamen als Argument oder mit einer Datei als Standardeingabe aufgerufen, liest und führt es ein Skript von dieser Datei aus.

Eine zweite Möglichkeit zum Starten des Python-Interpreters ist `python -c Befehl [arg] ...`, wodurch die Anweisung(en) in diesem Befehl ausgeführt werden,

analog zur `-c`-Option der Shell. Da Python-Anweisungen oft Leerzeichen oder sonstige Zeichen enthält, die von der Shell besonders behandelt werden, sollte man den kompletten Befehl in einfache Anführungszeichen setzen.

Einige Python-Module sind auch als Skripte nützlich und können mit `python -m Modul [arg] ...` aufgerufen werden. Dadurch wird der Quelltext von Modul ausgeführt, so als hätte man den vollständigen Namen in die Kommandozeile eingegeben.

Achtung: Es gibt einen Unterschied zwischen `python Datei` und `python < Datei`! Im zweiten Fall werden Eingabeanfragen des Programms, wie beispielsweise der Aufruf `sys.stdin.read()`, von Datei erledigt. Da diese Datei aber schon vom Parser bis zum Ende gelesen wurde, bevor mit der Ausführung begonnen wird, trifft das Programm sofort auf ein End-Of-File. In ersterem Fall passiert das, was man normalerweise erwartet: Die Eingabeanfragen werden durch diejenige Datei oder das Gerät erledigt, die bzw. das als Standardeingabe zur Verfügung steht.

Wenn eine Skriptdatei verwendet wird, ist es oft hilfreich, das Skript auszuführen und danach in den interaktiven Modus zu wechseln. Dies erreicht man durch die Option `-i` vor dem Skript.

2.1.1. Übergabe von Argumenten

Werden dem Interpreter ein Skriptname und zusätzliche Argumente übergeben, dann werden diese in eine Liste von Zeichenketten gewandelt und an die Variable `argv` im `sys` Modul übergeben. Zugriff darauf erhält man mittels `import sys`. Wenn kein Skript und keine Argumente übergeben wurden, dann ist `sys.argv[0]` eine leere Zeichenkette. Wenn der Skriptname als `'-'` angegeben ist (das entspricht der Standardeingabe), dann wird `sys.argv[0]` auf `'-'` gesetzt. Wird `-c` Befehl verwendet, dann erhält `sys.argv[0]` den Wert `'-c'`, bei Verwendung von `-m` Modul den vollständigen Namen des gefundenen Moduls. Optionen, die nach `-c` Befehl oder `-m` Modul angegeben werden, werden nicht vom Python-Interpreter verarbeitet, sondern werden als Werte an `sys.argv` übergeben.

2.1.2. Interaktiver Modus

Wenn Befehle von einem tty (in der Regel wird das eine Konsole sein) gelesen werden, spricht man vom interaktiven Modus des Interpreters. In diesem Modus wartet der Interpreter mit der primären Eingabeaufforderung, die normalerweise aus drei größer-als-Zeichen besteht (`>>>`), auf Eingaben des Anwenders. Nach Fortsetzungszeilen zeigt der Interpreter die sekundäre Eingabeaufforderung, das sind normalerweise drei Punkte (`...`). Außerdem zeigt der Interpreter nach dem Start zunächst einen kurzen Informationstext an,

der unter anderem die Versionsnummer des Interpreters und einen Hinweis zum Urheberrecht enthält.

```
$ python3.3
Python 3.3 (py3k, Apr 1 2010, 13:37:42)
[GCC 4.4.1] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Fortsetzungszeilen ergeben sich, wenn mehrzeilige Konstrukte eingegeben werden, wie zum Beispiel bei der folgenden `if`-Anweisung:

```
>>> the_world_is_flat = True
>>> if the_world_is_flat:
...     print("Be careful not to fall off!")
...
Be careful not to fall off!
```

2.2. Der Interpreter und seine Umgebung

2.2.1. Fehlerbehandlung

Tritt ein Fehler auf, dann zeigt der Interpreter eine Fehlermeldung mit einem Verlaufsbericht (Stacktrace) an. Im interaktiven Modus kehrt er dann zurück zur primären Eingabeaufforderung. Wenn die Eingabe von einer Datei kam, beendet er sich nach der Ausgabe des Fehlerberichts mit einem Rückgabewert ungleich Null. Ausnahmen (Exceptions), die in einem `try-except`-Block verarbeitet werden, gelten in diesem Zusammenhang nicht als Ausnahmen. Manche Fehler führen zum sofortigen Abbruch des Interpreters mit einem Rückgabewert ungleich Null. Dies gilt etwa bei internen Inkonsistenzen oder Speichermangel. Alle Fehlermeldungen werden in den Standardfehlerausgabestrom, gewöhnliche Ausgaben von ausgeführten Befehlen wird in die Standardausgabe geschrieben.

Die Eingabe des Interrupt-Zeichens (normalerweise `Strg-C` oder ENTF) bei der primären oder sekundären Eingabeaufforderung bricht die Eingabe ab und kehrt zur primären Eingabeaufforderung zurück. [2] Ein Interrupt während einer Befehlsausführung verursacht eine `KeyboardInterrupt`-Ausnahme, die durch eine `try`-Anweisung behandelt werden kann.

2.2.2. Ausführbare Python-Skripte

Auf BSD-ähnlichen Unixsystemen kann ein Pythonskript - ähnlich einem Shellskript - direkt ausführbar gemacht werden, indem man folgende Zeile (shebang) an den Anfang des Skripts schreibt

```
#!/usr/bin/env python3.3
```

Dabei wird vorausgesetzt, dass sich der Pfad zum Interpreter im `PATH` des Benutzers befindet. Die `#!` müssen die ersten zwei Zeichen der Datei sein. Auf manchen Plattformen

muss diese erste Zeile mit einem unixoiden Zeilenende (`'\n'`) enden und nicht mit einem Windows-Zeilenende (`'\r\n'`). Hinweis: Die Raute `'#'` dient in Python dazu, einen Kommentar zu beginnen.

Einem solchen Skript können dann Ausführungsrechte mit Hilfe des Befehls **chmod** verliehen werden:

```
$ chmod +x myscript.py
```

Auf Windowssystemen gibt es den Begriff der "Ausführungsrechte" nicht. Das Python-Installationsprogramm verknüpft automatisch `.py`-Dateien mit `python.exe`, sodass ein Doppelklick auf eine Python-Datei diese als Skript ausführt. Die Dateinamenserweiterung kann auch `.pyw` lauten, in diesem Fall wird das normalerweise auftauchende Konsolenfenster unterdrückt.

2.2.3. Kodierung von Quellcode

Standardmäßig werden Python-Quelltextdateien als in UTF-8 kodiert behandelt. In dieser Kodierung können die Zeichen der meisten Sprachen gleichzeitig in Stringliteralen, Bezeichnern und Kommentaren verwendet werden. Die Standardbibliothek verwendet allerdings nur ASCII-Zeichen für Bezeichner - eine Konvention, der jeder portable Code folgen sollte. Um alle diese Zeichen korrekt darzustellen, muss ein Editor erkennen, dass die Datei UTF-8 kodiert ist und einen Font benutzen, der alle Zeichen der Datei unterstützt.

Will man eine andere Kodierung als UTF-8 für eine Quelltextdatei verwenden, dann muss unmittelbar unterhalb der `#!` Zeile eine weitere, spezielle Kommentarzeile eingefügt werden, durch die die Kodierung festgelegt wird

```
# -*- coding: Kodierung -*-
```

Mit dieser Angabe wird alles in der Quelltextdatei so behandelt, als hätte es die Kodierung Kodierung an Stelle von UTF-8. Die Liste der möglichen Kodierungen findet man in der Python Library Reference, in der Sektion zu **`codecs`**.

Wenn ein Editor beispielsweise keine UTF-8 kodierten Dateien unterstützt und auf die Benutzung einer anderen Kodierung besteht, sagen wir mal Windows-1252, kann man durch folgende Kodierungszeile

```
# -*- coding: cp-1252 -*-
```

immernoch alle Zeichen des Windows-1252 Zeichensatzes im Quelltext verwenden. Dieser spezielle Kodierungskommentar muss in der ersten oder zweiten Zeile der Datei stehen.

2.2.4. Die interaktive Startup-Datei

Wenn Python interaktiv genutzt wird, ist es gelegentlich hilfreich, bei jedem Start des Interpreters einige Standardbefehle automatisch auszuführen. Das lässt sich erreichen, indem man eine Umgebungsvariable namens `PYTHONSTARTUP` erstellt, die auf eine Datei mit den Startup-Befehlen verweist. Dies ist vergleichbar mit der `.profile`-Datei von Unixshells.

Diese Datei wird nur in interaktiven Sitzungen gelesen. Wenn der Interpreter ein Skript ausführt oder `/dev/tty` explizit als Quelle angegeben wird - was ansonsten einer interaktiven Sitzung entspricht -, wird die Startup-Datei nicht berücksichtigt. Ausgeführt wird sie im selben Namensraum wie interaktive Befehle, so dass Objekte, die in der Startup-Datei definiert oder importiert werden, ohne Qualifizierung in der interaktiven Sitzung genutzt werden können. Auch die Eingabeaufforderungen `sys.ps1` und `sys.ps2` lassen sich in dieser Datei festlegen.

Sollen noch weitere Startup-Dateien aus dem aktuellen Verzeichnis gelesen werden, dann lässt sich dies durch Code
wie `if os.path.isfile('.pythonrc.py'): exec(open('.pythonrc.py').read())` in der globalen Datei erreichen. Soll die Startup-Datei in einem Skript verwendet werden, muss das explizit in diesem Skript geschehen:

```
import os
filename = os.environ.get('PYTHONSTARTUP')
if filename and os.path.isfile(filename):
    exec(open(filename).read())
```

2.2.5. Die Customization Module

Python bietet zwei Haken, um es anzupassen: `sitecustomize` und `usercustomize`. Um es auszuprobieren, musst du zuerst den Ort deines Benutzer `site-packages` Ordners herausfinden. Starte Python und gib dies ein:

```
>>> import site
>>> site.getusersitepackages()
'/home/user/.local/lib/python3.2/site-packages'
```

Dort kannst du eine Datei namens `usercustomize.py` anlegen und alles gewünschte eingeben. Es wird jeden Aufruf von Python beeinflussen, es sei denn der Aufruf enthält die Option `-s`, um den automatischen Import zu verhindern.

`sitecustomize` funktioniert genauso, aber es wird typischerweise von einem Administrator im globalen `site-packages` Ordner erstellt und vor `usercustomize` importiert. Mehr dazu in der Dokumentation des `site`-Moduls.

Fußnoten

[1] Unter Unix wird der Python 3.1 Interpreter nicht standardmäßig als ausführbare Datei

namens `python` installiert, damit es nicht zu einer Kollision mit einer gleichzeitig installierten Python-2.x-Version kommt.

[2] Ein Problem mit dem GNU-readline-Paket kann dies verhindern.