Python - Übungsblatt 7

Gerald Senarclens de Grancy

March 6, 2007

Übung 1 - Objekte speichern

Oft will man ein wichtigeres Objekt, zum Beispiel ein grösseres Dictionary, welches eine Datenbank enthält, als ganzes in eine Datei speichern. In Python gibt es dazu mehrere Möglichkeiten, wovon wir uns das `pickle` Modul ansehen wollen. Dabei reicht es, eine Datei zu öffnen und das Objekt mit
`pickle.dump(myObject, file)`
darin zu speichern. Will man ein Objekt laden, so muss man die Datei, in der es sich befindet für den Lesezugriff öffnen und das Objekt mit
`object = pickle.load(file)` laden. Um das Modul kurz zu testen, öffne sowohl Idle als auch eine Python shell in der regulïen Eingabeaufforderung. In beiden importiere das `human` Modul. In der Eingabeaufforderung lege nun mehrere `human` und `male` Objekte an und lade diese in Idle. Beachte dabei, dass beide Shells im gleichen Verzeichnis laufen müssen (`os.getcwd()`), wenn du keine absoluten Pfadangaben zu den Dateien verwenden willst.

Übung 2 - Exceptions

Exceptions (Ausnahmen) dienen dazu, unter gewissen Umständen (meist Fehlerzuständen) die Kontrolle über das Programm zur Weiterbehandlung an eine andere Ebene des Programms weiterzureichen. Python stellt ein ausgeklügeltes System zur Behandlung von Exceptions zur Verfügung, indem man sowohl vorgefertigte Exceptions verwenden als auch einfach eigene Exceptions kreieren kann. Versuche anhand der folgenden, erklärenden Beispiele unsere Klasse `Human` mit einigen Fehlerabsicherungen zu versehen.

```
myError = a very simple exception
def problemFunction():
   raise myError, additional description of the exception
try:
   problemFunction()
except myError, data:
   print Exception caught:, data
else:
   print no exception occured
```

```
try:
   raise new Exception
finally:
   print doing cleanup
```

Übung 3 - Wichtige Module

Sys

Mit dem Sys-Modul kann man auf Objekte und Funktionen zugreifen, die mit dem Python Interpreter in Zusammenhang stehen, so zum Beispiel der Pfad, in dem Nach Modulen gesucht wird. Das für den täglichen gebrauch wichtigste Element von sys ist vermutlich argv, eine Liste mit Argumenten, die von der Kommandozeile an das Modul übergeben werden.

Schreibe ein Modul, welches bei Aufruf von der Kommandozeile Daten zu einem `human` Objekt ausgibt, welches es aus einer Datei liest, die ihm beim Aufrufen als Argument übergeben wurde.

OS

Das OS-Modul dient speziell zur Interaktion mit dem zugrundeliegenden Betriebssystem. Dabei lassen sich sämtliche POSIX Funktionen ausführen, es steht ein Zugriff auf die Shell des Betriebssystems zur Verfügung (damit kann man jegliches im OS installierte Programm aufrufen) und es ist essentiell für plattformunabhängige Programmierung. Ein paar wichtige Elemente sind:
`os.sep` (das Verzeichnistrennzeichen)
`os.linesep` (das Zeilenumbruchzeichen)
`os.system(shell command)`
`os.getcwd()` (gibt das aktuelle Verzeichnis zurück)
`os.chdir(path)` (wechselt das aktuelle Verzeichnis)

String

Mit dem String Modul lassen sich Zeichenketten manipulieren. Die angebotenen Funktionen reichen für viele Benutzer aus, wenn jedoch reguläre Ausdrücke bei der Suche benötigt werden, so sollten andere Module verwendet werden. Einige wichtige Funktionen und Konstanten sind:
`string.{digits, hexdigits, letters, lowercase,...}`
`atof(s)` konvertiert einen String in ein float (wie `float()`
`atoi(s)` konvertiert einen String in ein int (wie `int()`
`atol(s)` konvertiert einen String in ein long (wie `long()`
`s.find(sub [, start [, end]])`, `s.rfind(...)`, `s.index(...)`,... Suchfunktionen
`s.split([sep [, maxsplit]])`, `join(x [, sep])` teilt bzw. vereint Strings
`s.replace(old, new [, maxsplit])` ersetzt `old` durch `new` im Rückgabestring
...

Weitere Module

Bereits in den obigen Modulen sind nur wenige der wichtigsten Funktionen der Module aufgelistet, da eine vollständige Aufzählung den Rahmen sprengen und vermutlich nur für Verwirrung sorgen würde. Ausser den oben genannten Modulen werden mit Python selbst noch viele weitere mitgeliefert, darunter das `Re` Modul für reguläre Ausdrücke, `os.path`, `pickle`, `shelve`,... Da die Menge der Module und deren Umfang sehr groß ist, ist eine erschöpfende Behandlung dieser nicht sinnvoll. Besser ist es hier, sich jeweils die Module anzusehen, die man gerade braucht. Der beste Einstieg in ein Modul ist wohl

```
import modul
help (modul)
```

Ausser all den Modulen, die direkt in Python enthalten sind, gibt es eine nahezu unendlich lange Liste von Modulen die durch Dritte angeboten werden. Darunter befinden sich sowohl sehr viele Open Source, als auch viele kostenfreie und kostenpflichtige Module. Diese werden in der Regel als installierbare Packete für verschiedene Betriebssysteme angeboten. Eines dieser Open Source Packete ist Biopython. Module können aufeinander aufbauen und daher Abhängigkeiten besitzen, was auch bei Biopython der Fall ist. Damit Biopython korrekt funktionieren kann, benötigt man unter anderem auch die Egenix und Numeric Python.