Python - Übungsblatt 4

Gerald Senarclens de Grancy

March 2, 2007

Übung 1 - Ein erstes Modul erstellen

Im Folgenden werden wir ein Modul `convert.py` erstellen, das einige Umrechnungen zwischen verschiedenen Einheiten erledigen soll. Dazu definieren wir in dem Modul einige Beispielfunktionen und Konstanten.

EUR⇔ATS

Erstelle zwei Funktionen, die je ein Argument (Euro bzw. Schilling) haben und die jeweis andere Währung zurückgeben. Der fixe Umrechnungskurs für EUR 1 beträgt ATS 13,7603. Die Definitionszeilen sollen wie folgt aussehen

```
def eur2ats(eur):
def ats2eur(ats):
```

Celsius⇔Fahrenheit

Erstelle ein weiteres Funktionspaar, diesmal zur Umrechnung zwischen der kontinentaleuropäischen Celsius- und der amerikanischen Fahrenheitskala. Die Umrechnungsgleichungen lauten `C = (F - 32) / 1.8` bzw. `F = (C * 1.8) + 32` resprektive. Die Definitionszeilen der Funktionen sollen so aussehen:

```
def celsius2fahrenheit(celsius):
def fahrenheit2celsius(fahrenheit):
```

Celsius⇔Kelvin

Erstelle ein weiteres Funktionspaar zur Umrechnung zwischen der Celsius- und der Kelvin Temperaturskala. Die Umrechnungsgleichungen lauten `C = K - 273.15` bzw. `K = C + 273.15` resprektive. Die Definitionszeilen der Funktionen sollen so aussehen:

```
def celsius2kelvin(celsius):
def kelvin2celsius(kelvin):
```

Konstanten

Erweitere dein Modul um einige zum Thema gehörige nützliche Konstanten[1] (einfach einer Variable im obersten Leven des Moduls einen Wert zufügen. Definiere auf diese Weise
`abs_zero_celsius=-273.15`
`abs_zero_kelvin=0`
`abs_zero_fahrenheit=-459.67`

Übung 2 - Das Modul verwenden

Konsole und PYTHONPATH

Starte den Python Interpreter in der Konsole und versuche das Modul convert.py (`import convert`) zu laden. Warum kannst du es nicht importieren?
...
Um ein Modul verwenden zu können, muss es sich in einem der in der Variable `PYTHONPATH` genannten Verzeichnisse befinden. Diese Variable enthält unter anderem standardmässig das aktuelle Verzeichnis. Beende daher den Interpreter, wechsle in das Verzeichnis, in welchem sich dein Modul befindet und versuche es dort erneut in den Interpreter zu importieren.

Nachdem es kompliziert (wenn nicht sogar unmöglich) ist, jedes mal in das Verzeihnis des importen Moduls zu wechseln, verändere nun die Variable `PYTHONPATH` in deinem Betriebssystem indem du sie um ein Verzeichnis, in welchem du alle deine Module ablegst, erweiterst. Danach sollte der Import dieser Module in jedem Verzeichnis ohne Schwierigkeiten möglich sein.

Verwendung des Moduls in einem anderen Modul

Erstelle nun eine neue Moduldatei, die das eben erstellte Modul importiert (sollte nach korrekter Konfiguration der Variable `PYTHONPATH` kein Problem sein (Vorsicht, mache Editoren verwenden leider nicht die Variable des Betriebssystems sondern müssen gesondert konfiguriert werden)). Verwende nun jede der im Modul definierten Funktionen mindestens einmal und gib auch jede der definierten Konstanten aus.

Verfeinern des Moduls

Hilfe

Von anderen Modulen ist man es gewöhnt, dass sie einem eine nützliche Hilfe in der Konsole anbieten. Diese Hilfe wird von Python auch für selbst verfasste Module automatisch erstellt. Rufe daher in der Konsole nach dem Import des Moduls den Befehlt `help(convert)` auf und erkläre, warum keine so ausführliche Erklärung wie bei anderen Modulen angeboten wird.
...

[1] Python kennt das Konzept von Konstanten nur eingeschränkt. Bei anderen Programmiersprachen versteht man unter einer Konstanten einen nach der Definition nicht mehr veränderbaren Wert, was aber bei Python in der Form nicht möglich ist.

Genau - woher sollte Python wissen, was jede unserer Funktionen und das Modul insgesamt erledigen bzw. was unsere Intention ist? Wir müssen es also um sogenannte Docstrings (starten und enden mit """) erweitern. Nachdem dies erledigt ist, sollten wir das Modul erneut importieren (am besten mit `reload(convert)` bevor wie die aktualisierte Hilfe für unser Modul anzeigen können.

Eine Testfunktion

Abschliessend sollten wir noch eine Funktion erstellen, die das Modul testet. Diese Funktion könnte zum Beispiel den Namen `testme()` haben und sollte jede der im Modul befindlichen Funktionen durch verschiedene Aufrufe testen (es empfiehlt sich, insbesondere "kritische" Werte für die Argumente zu probieren). Dabei ist es besonders wichtig, dass die Testfunktion ausreichen Output gibt (verwende erkleuternde `print` Statements. Vergiss nicht, auch diese Funktion mit einem Docstring zu versehen.

Der Benutzer des Moduls hat nun die Möglichkeit nach Import des Moduls dieses mittels der Funktion zu testen. Bei Python ist es aber üblich, dass manche Module nicht nur importiert werden können, sondern auch von der Kommandozeile aus aufgerufen werden. Da auf dem Toplevel des Moduls bisher keine Funktion gestartet wird, wuerde aber nichts passieren. Bei Modulen, die hauptsächlich für den Import gedacht sind, ist es eventuell eine gute Idee, die Testfunktion beim Aufruf durch die Kommandozeile automatisch zu starten, aber jedoch nicht beim Import durch ein anderes Modul oder den Python Interpreter. Die kann man mit folgendem Code erreichen, der überprüft, ob das Modul direkt aus dem Betriebssystem gestartet wird (es erhält in diesem Fall den Namen `__main__` von Python)

```
if __name__ == "__main__":
   testme()
```