Programmierung mit Python für Einsteiger:: Kapitel 5 - Ausnahmen, Module, Dateien

Inhaltsverzeichnis

Ausnahmebehandlung - Motivation

- Im „Nachtbetrieb“ werden „viele“ Datensätze bearbeitet. Was passiert, wenn beim einem Datensatz ein Fehler auftritt (z.B. „Divsion durch Null“)?
- Das Programm bricht ab (siehe ❗ `ex1.py`)!
- Wünschenswert wäre aber, dass der Datensatz ignoriert wird, und das Programm mit der Verarbeitung des nächsten Datensatzes weitermacht.
- Wie kann man das erreichen? Siehe ❗ `ex2.py`.
- Nachteile:

- Der Code wird aufgebläht. Die eigentliche Fachlichkeit ist nicht mehr zu erkennen.
- Man muss an jeder Stelle im Code überlegen, was im Falle eines unerwarteten Ereignisses zu tun ist.

Ausnahmen fangen

- Bessere Lösung: Siehe ❗ `ex3.py`.
- Ausnahmen, die in einem *Codeblock* geworfen werden (`try`), können „gefangen“ werden.
- In einem `except`-Block wird definiert, was zu tun ist, wenn die Ausnahme auftritt.
- Ausführung:
  - Die Anweisungen zwischen `try` und `except` werden ausgeführt.
  - Tritt keine Ausnahme auf, wird der `except`-Block übersprungen.
  - Tritt eine Ausnahme auf, so wird sofort bei deren Auftreten in den `except`-Block gesprungen.

Ausnahmetypen

- Mit `except Exception` (wie im Beispiel) werden alle Arten von Ausnahmen gefangen (was man aber nicht immer will).
- Es können auch „Unterarten“ geworfen und (verschieden) behandelt werden.
- Beispiel: `int("Hallo")` liefert einen ValueError.
- Man hätte also auch `except ValueError` schreiben können.
- Tritt eine Ausnahme auf, die nicht abgefangen wird, wird die Ausnahme weiter „nach oben“ geworfen.
- Beispiele:
  - ❗ `ex4.py`: Abfangen von zwei Ausnahmen
  - ❗ `ex5.py`: Division durch 0 wird nicht gefangen

Ausnahmen werfen

- Wenn im eigenen Programm eine Situation auftritt, die an dieser Stelle nicht erwartet wurde, kann man auch selbst eine Ausnahme werfen (Bsp: als Entwickler der `int`-Funktion):

```
    raise Exception(“Fehler: ...“)
```

- Beispiel: ❗ `ex6.py`

- Statt eine Ausnahme vom Typ `Exception` zu werfen, kann man auch eigene Ausnahmetypen definieren (was i.a. auch besser ist). Das kommt aber später (neue Klasse im Sinne der Objektorientierten Programmierung)!

Ausnahmen werfen - Anmerkungen

- **Empfehlung**: Ausnahmen nur werfen, wenn ein unerwarteter Fall auftritt, nicht, um (erwartete) Programmlogik zu implementieren.
- Es ist eine (manchmal schwierige) Entwurfsentscheidung, wann man Ausnahmen benutzt und wann eine „normale“ Behandlung (z.B. via if – Abfrage).

Ausnahmen - Übung / Fallbeispiel

- Programm zur Berechnung eines Durchschnittsverbrauch: ❗ `ex7.py`.
- Der „gerade“ (fehlerlose) Fall funktioniert. Das Programm ist aber nicht „stabil“. Programmabbruch bei Fehler.
- Übung 1: Bei einem Fehler soll eine entsprechende Ausgabe erfolgen. Es soll der Durchschnitt der übrigen Monate berechnet werden: ❗ `ex8.py`.
- Übung 2: Einträge mit einer ungültigen Monatsbezeichnung (d.h. nicht in der Liste `monate`) sollen nicht berücksichtigt werden. Es soll eine entsprechende Ausgabe erfolgen: ❗ `ex9.py`.
- Übung 3: Wie sähe eine Lösung ohne Ausnahmen aus?
- Bemerkung: In der Praxis würde man die Ausgaben in eine „Log-Datei“ schreiben.

❓ Übung 12 - Ausnahmen und röm. Zahlen

- Was passiert, wenn Sie bei der aktuellen Lösung eine Zeichenkette angeben, die keiner römischen Zahl entspricht?
- Schreiben Sie ein Programm, das in einer Endlosschleife eine Eingabe entgegen nimmt:
  - Bei der Eingabe von `"Ende"` soll das Programm beendet werden.
  - Bei einer fehlerhaften Eingabe soll eine entsprechende Meldung ausgegeben werden und mit der nächsten Eingabe fortgefahren werden.
- siehe Ordner `12_uebung_roemische_zahl_ausnahmen`

Module - Motivation

- Motivation von Funktionen: Wiederverwendung

  - des eigenen Codes
  - aber auch von „Fremd-Code“.
- Wie kann man nun Funktionen, die in einem Programm definiert sind, in anderen Programmen (wieder-) verwenden?
- Lösung: Import der Python-Datei, in der die Funktionen definiert wurden. In diesem Fall nennt man die Python-Datei „Modul“.
- Beispiel: ❗ `mod1.py` , `mod2.py`
- In einem Modul kann man Funktionen zu einem Thema bündeln (z.B. mathematische Funktionen, Funktionen zur String-Verarbeitung, Funktionen für Dateibehandlung).
- Module stellen also eine weitere Ebene der Strukturierung des Quellcodes dar.

Module - Anwendung

- Import eines Moduls mit der Import-Anweisung (Dateiname der Python-Datei ohne Endung .py)

```
import mod1
```

- Die Funktionen und globalen Variablen, die in der importierten Python-Datei definiert wurden, können dann durch voranstellen des Modulnamens verwendet werden:

```
mod1.f1()
```

Module - Suchpfad

- Wo werden Module gesucht, die per Import geladen werden sollen?
  - im aktuellen Verzeichnis
  - im Standardpfad der Python-Installation (z.B. math)
  - Umgebungsvariable PYTHONPATH
- Abkürzung:

```
import mod as m
m.f1() # statt mod.f1()
```

- Es gibt noch andere Möglichkeiten zum Import, die hier nicht besprochen werden sollen (s. https://www.python-kurs.eu/python3_modularisierung.php)

Module - Import

- Was passiert eigentlich genau bei dem Import eines Moduls (also einer Python-Datei)?
- Der Interpreter führt die Datei aus, d.h.
  - Funktionsdefinitionen und globale Variablen sind danach an der Aufrufstelle bekannt – mit dem Modulnamen als Präfix (das ist der eigentliche Sinn des Imports).
  - „Normale Anweisungen“ werden aber auch ausgeführt.
- Letzteres ist normalerweise nicht gewünscht. Was kann man dagegen tun?
- Die Anweisungen, die nicht in Funktionen stehen und die nicht beim Import ausgeführt werden sollen, hinter if-Block:

```
    if __name__ == “__main__“:
```

- Beispiel: ❗ `mod3.py`, `mod4.py`.

Pakete

- Wie kann man zusammengehörige Module gruppieren?
  - Zusammenfassung zu einem Verzeichnis!
  - Das nennt man dann im Python-Sprech „Paket“
  - Ein Paket ist also zunächst nichts anderes als ein Verzeichnis.
- Eine Besonderheit gibt es dann doch: Datei `*init*.py` im Verzeichnis (s. später).
- Pakete werden dort gesucht, wo auch Module gesucht werden.
- Pakete sind also nach Funktionen und Modulen eine weitere Ebene der Stukturieren des Quellcodes

---

Verwendung von Paketen

- Wie verwendet man Module aus einem Paket?
  - Import des Moduls über den vollständigen Pfad (vgl. Pfad im Betriebssystem, aber Trennung mit Punkt statt /). Beispiel: ❗ `pak1.py`
  - Import des Moduls, wenn im Paket die Module via der Datei `*init*.py` eingeladen werden: Beispiel: ❗ `pak2.py`
- Wie ein Verzeichnis Dateien und weitere Verzeichnisse enthalten kann, kann ein Paket Module und auch weitere Pakete enthalten.

Pakete - Import

Was passiert beim Import eines Pakets?

- Beim Import eines Pakets wird die Datei `__init__.py` in dem entsprechenden Ordner ausgeführt.
- Dort kann man also die Importe der Module durchführen.

Module und Pakete - Zusammenfassung

Modul

- Ein Modul ist eine Python-Datei.
- Beim Import eines Moduls wird die Datei ausgeführt, wodurch Funktionen und globale Variable an der Aufrufstelle bekannt sind.

Paket

- Ein Paket ist ein Verzeichnis mit der Datei `__init__.py`
- Beim Import wird diese Datei ausgeführt.
- Sie kann also dazu verwendet werden, die Module (=Python-Dateien) in dem Paket (=Verzeichnis) zu importieren. Muss man aber nicht.

❓ Übung 13 - Module

- Die Funktionen zur Bearbeitung römischer Zahlen (in dem Fall die Umwandlung in eine Dezimalzahl) seien in einem Modul `roemische_zahlen` definiert.
- Schreiben Sie ein Programm roemische_zahlen_test, das eine römische Zahl einliest und mit Hilfe der Funktion des Moduls in eine römische Zahl umwandelt.
- siehe Ordner `uebungen/13_uebung_roemische_zahl_modul`

Dateien

- Die im Programm verarbeiteten Daten sind nach der Programmausführung weg.
- Möglichkeit der dauerhaften („persistenten“) Speicherung: Dateien
- Datei: Menge von logisch zusammenhängenden und meist sequentiell geordneten Daten, die auf einem Speichermedium dauerhaft gespeichert werden und mittels eines Bezeichners bzw. Namens wieder identifizierbar und damit ansprechbar sind.
- Eindimensionale Aneinanderreihung von Bits.

Lesen aus einer Datei

- Lesen aus einer Datei:

```
fobj = open("dateiname", "r")
for line in fobj:
    print(line)
fobj.close()
```

- Beispiel: ❗ `datei1.py`
- Datei wird zeilenweise gelesen als Zeichenkette (String).
- Jede Zeile enthält am Ende den Zeilenumbruch
- Den muss man evtl. entfernen (`line.strip()`)
- Beispiel: ❗ `datei2.py`
- Die Datei muss am Ende „geschlossen“ werden.

Lesen aus einer Datei 2

- Möglicherweise muss man zu Weiterverarbeitung eine Typkonvertierung durchführen:
- Beispiel: ❗ `datei3.py`

Schreiben in eine Datei

- Analog werden Strings zeilenweise in eine Datei geschrieben:

```
fobj = open(“ausgabedatei“,“w“)
fobj.write(“Zeile1\n“)
fobj.write(“Zeile2\n“)
fobj.close
```

- Dabei ist `"\n"` ein Zeichen (!), nämlich das Zeichen für den Zeilenumbruch (für alle Betriebssysteme!).
- Das Zeilenumbruchszeichen muss explizit für jede Zeile geschrieben werden.
- Gegebenenfalls müssen also die zu schreibenden Daten in Strings umgewandelt werden (plus Zeilenumbruch).
- Beispiel: ❗ `datei4.py`

Funktionen `read` und `readlines`

- Lesen aller Zeilen in eine Liste:

```
list_of_lines = open("meinedatei","r").readlines()
```

- Lesen aller Zeilen in einen einzigen String:

```
string_with_lines = open("meinedatei","r").read()
```

❓ Übung 14 - Dateien (Namen) [Kle]

- Gegeben sei eine Datei, in der jeweils abwechselnd ein Vorname und ein Nachname in einer Zeile steht, also beispielsweise so:

```
Fred
Miller
Eve
Turner
Steve
Baker
```

  Diese Datei soll in eine andere Datei überführt werden, in der pro Zeile jeweils ein Vorname und ein Nachname steht.

- siehe Ordner `uebungen/14_uebung_dateien_namen`

❓ Übung 15 - Dateien (röm. Zahl)

- Schreiben Sie ein Programm, das aus einer Datei römische Zahlen ausliest (eine Zahl pro Zeile) und den zugehörigen Dezimalwert in eine andere Datei schreibt (wiederum eine Zahl pro Zeile).
- siehe Ordner `uebungen/15_uebung_roemische_zahl_datei`

Referenzen

- [Kle] Bernd Klein, Einführung in Python 3