Ausnahmebehandlung

Try except finally

Fehlerbehandlung und Ausnahmebehandlung

Eine Ausnahme (exception) ist eine Ausnahmesituation (Fehler), die sich während der Ausführung eines Programmes einstellt.

Unter einer Ausnahmebehandlung (exception handling) versteht man ein Verfahren, die Zustände, die während dieser Situation herrschen, an andere Programmebenen weiterzuleiten.

Dadurch ist es möglich, per Code einen Fehlerzustand gegebenenfalls zu "reparieren", um anschließend das Programm weiter auszuführen.

Ansonsten würden solche Fehlerzustände in der Regel zu einem Abbruch des Programmes führen.

Umsetzung

Die Realisierung der Ausnahmebehandlung sieht meist so aus, dass automatisch, wenn eine Ausnahmesituation auftritt, Informationen und Zustände gespeichert werden, die zum Zeitpunkt der Ausnahme bzw. vor der Ausnahme geherrscht hatten.

An dieser Stelle wird der normale Programmfluss unterbrochen und ein spezielles Code-Fragment ausgeführt, der auch Exception-Handler genannt wird.

Bedingt durch die Art des Fehlers ("Division durch 0", "Datei-Fehler", usw.) kann das spezielle Code-Fragment darauf reagieren. Danach kann wieder der normale Programmfluss aufgenommen werden mit den vorher gespeicherten Informationen und Zuständen.

Schema

```
try:
    # kritischer Code, der unter Umständen einen Fehler provoziert
except:
    # hier weiterarbeiten
else:
    #wird ausgeführt, wenn keine Exception aufgetreten ist
finally:
    # das in jedem Fall ausführen, auch wenn
    # die Exception nicht aufgefangen wird
```

Look before you leap (LBYL)

Herkömmlicher Ansatz für das Zero-Division-Problem:

```
if x !=  0:
    z = y / x
else:
   z = None
```

It's easier to ask forgiveness than *it is* to get permission (EAFP)

```
try:
    z = 3 / x
except:
    print("Division durch Null ist nicht erlaubt!")
    z = None
```

Spezifieren der Ausnahme

Obwohl es möglich ist, alle Fehler in einer einzigen unbenannten except-Anweisung abzufangen, wird davon abgeraten.

```
try:
    do_something()
except:
    # hier alle Fehler auffangen, die im try-Block passieren könnten
```

Besser ist es, die Ausnahme zu spezifizieren:

```
try:
    p = 3/0
except ZeroDivisionError:
  print(„Division durch Null nicht möglich“)
```

Finally

Das Finally-Statement sieht auf den ersten Blick unnötig aus. Es wird immer ausgeführt, egal ob die Exception aufgefangen wurde oder nicht. Nützlich zum Beispiel bei Dateioperationen:

```
try:
    f = open("data.txt", "r")
    f.write("Wir haben f aber nur im Lesemodus geöffnet!")
except NameError:
    print(„nur wenn ein Name-Error auftritt, wird das ausgeführt")
finally:
    print("egal was passiert, wir schließen das File aber wieder")
    f.close()
```

Beispiel: Fehler bei Division durch Null

p = 4 / 0

⇒Der python Interpreter gibt uns einen ZeroDivisionError aus. Wir mussten uns an dieser Stelle anders behelfen, zum Beispiel mit einer bedingten Abfrage, bevor die Division durchgeführt wird (!=0).

Mit Exceptions können wir dieses Problem so lösen:

Beispiel:

```
try:
    x = 4 / 0
except ZeroDivisionError:
    print("Da ist wohl ein Fehler aufgetreten.")
```

Das Programm bricht nicht ab, es läuft weiter.

Welche Exceptions gibt es?

Es gibt neben zahllosen built-in Exceptions wie ValueError oder NameError noch die modulspezifischen Exceptions (z.b. Numpy) und die user-erstellten Exceptions.

Eine Liste aller build-in-Exceptions findet sich in der Python Dokumentation:

https://docs.python.org/3/library/exceptions.html

Eine Exception werfen (raise)

Es ist möglich, selbst Exceptions auszulösen.

```
x = 10
if x > 5:
    raise ValueError('Eigene Fehlermeldung hier angeben')
```

Auch diese können dann mit try except aufgefangen werden.