Ausnahmen-Behandlung

Prof. Dr. Rüdiger Weis

TFH Berlin

Wintersemester 2008/2009

1. try..except..else
2. try..finally
3. raise
4. assert

Ausnahmen in Python

- Trennung von funktionalem Code und Fehlerbehandlung.
- Gute Sprachintegration
- Einfache Verwendbarkeit
- Ähnlich Java Exceptions
- Ausnahme Objekte

Why was Python created in the first place?

`http://www.python.org/doc/faq/general/`

1.1.2 Why was Python created in the first place?

Guido van Rossum:

> *'I was working in the Amoeba distributed operating system group at CWI. We needed a better way to do system administration than by writing either C programs or Bourne shell scripts, since Amoeba had its own system call interface which wasn't easily accessible from the Bourne shell. My experience with error handling in Amoeba made me acutely aware of the importance of exceptions as a programming language feature.'*

try..except..else

try..except..else

```
try:
   <anweisungen>
except [<name> [, <data>]]:
   <anweisungen>
[else:
   <anweisungen>]
```

- `except:` fängt alle Ausnahmen

try..except..else

- Falls innerhalb des Codeblocks nach der try Anweisung ein Ausnahme geworfen wird,
  - wird der erste passende except Block ausgeführt.
  - Falls kein passender except Block vorhanden ist, wird die Ausführung abgebrochen und die Ausnahme an die aufrufende Umgebung weitergeleitet.
- Nur wenn keine Ausnahme geworfen wurde, wird des else-Blocks ausführt

Beispiel ZeroDivisionError

```
>>> x=42/0
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
ZeroDivisionError: integer division or modulo by zero
>>> try:
...     x=42/0
... except ZeroDivisionError:
...     print('Bitte nicht durch 0 teilen.')
...
Bitte nicht durch 0 teilen.
```

Beispiel except

```
def dividiere(x,y): return(x/y)
tests=((42,2),(3,0),(4,5))
try:
    for a,b in tests:
        print('Division'),
        print(a,b),
        print('mit Resultat'),
        print dividiere(a,b)
except ZeroDivisionError:
    print('\n AUSNAHME: Bitte nicht durch 0 teilen.')
else: print('Alles unter Kontrolle')
print('\n Bye.')
```

Beispiel except Ausgabe

```
Division (42, 2) mit Resultat 21
Division (3, 0) mit Resultat
 AUSNAHME: Bitte nicht durch 0 teilen.

 Bye.
```

Beispiel else

```
def dividiere(x,y): return(x/y)
tests=((42,2),(3,1),(4,5))
try:
    for a,b in tests:
        print('Division'),
        print(a,b),
        print('mit Resultat'),
        print dividiere(a,b)
except ZeroDivisionError:
    print('\n AUSNAHME: Bitte nicht durch 0 teilen.')
else: print('Alles unter Kontrolle')
print('\n Bye.')
```

Beispiel else Ausgabe

```
Division (42, 2) mit Resultat 21
Division (3, 1) mit Resultat 3
Division (4, 5) mit Resultat 0
Alles unter Kontrolle

 Bye.
```

try...finally

try..finally

```
try:
    <anweisungen>
finally:
    <anweisungen>
```

- finally Block wird in jedem Fall ausgeführt ("Aufräumarbeiten").
- Bei einer Ausnahme, wird die Ausführung abgebrochen und die Ausnahme an die aufrufende Umgebung weitergeleitet.

Beispiel finally ohne

```
def dividiere(x,y): return(x/y)
tests=((42,2),(3,1),(4,5))
try:
    for a,b in tests:
        print('Division'),
        print(a,b),
        print('mit Resultat'),
        print dividiere(a,b)
finally:
    print('\n Aufraeumen...')
print('\n Bye.')
```

Beispiel finally Ausgabe ohne

```
Division (42, 2) mit Resultat 21
Division (3, 1) mit Resultat 3
Division (4, 5) mit Resultat 0

 Aufraeumen...

 Bye.
```

Beispiel finally mit

```
def dividiere(x,y): return(x/y)
tests=((42,2),(3,0),(4,5))
try:
    for a,b in tests:
        print('Division'),
        print(a,b),
        print('mit Resultat'),
        print dividiere(a,b)
finally:
    print('\n Aufraeumen...')
# Hinter try Block
print('\n Bye.')
```

Beispiel finally Ausgabe mit

```
Division (42, 2) mit Resultat 21
Division (3, 0) mit Resultat
 Aufraeumen...
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
  File "divfinally.py", line 8, in ?
    print dividiere(a,b)
  File "divfinally.py", line 1, in dividiere
    def dividiere(x,y): return(x/y)
ZeroDivisionError: integer division or modulo by zero
```

Vor Python 2.5: Entweder except..[else] oder finally

Vor Python 2.5: Entweder except..[else] oder finally

`try..finally` und `try..except..else` **nicht** kombinierbar.

Ab Python 2.5 (Relase: 19. September 2006) erlaubt

- PEP 341: Unified try/except/finally
- `http://docs.python.org/whatsnew/pep-341.html`

Python 2.5: Unified try/except/finally

http://www.python.org/dev/peps/pep-0341/

```
try:
    block-1 ...
except Exception1:
    handler-1 ...
except Exception2:
    handler-2 ...
else:
    else-block
finally:
    final-block
```

raise

raise

```
raise <name> [,<daten>]
```

- raise Schlüsselwort
- Wirft Ausnahme
- <name> String oder Ausnahme-Objekt
- <daten> Argument der Ausnahme

assert

assert

```
assert <bedingung> [, <data>]
```

- Zusicherung
- Wirft `AssertionError`

Zusicherung

```
assert <bedingung> [, <data>]
 # ist aequivalent zu

if __debug__:
   if not <bedingung>:
         raise AssertionError, <data>
```

- Globale Variable __debug__ per default gesetzt.
- -O setzt __debug__ = False

Beispiel: assert

```
>>> def f(x,y):
...     assert(y!=0)
...     return x/y
...
>>> f(8,2)
4
>>> f(2,0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in f
AssertionError
```

Der Plankalkül

Zusicherungen bereits in Plankalkül

- Erdacht in den Jahren 1942 bis 1946.
- Zuse, Konrad: Der Plankalkül. Gesellschaft für Mathematik und Datenverarbeitung. Nr. 63, BMBW - GMD - 63, 1972
- `http://de.wikipedia.org/w/index.php?title=Plankalk%C3%BCl`

©opyleft

©opyleft

- Erstellt mit Freier Software
- © Rüdiger Weis, Berlin 2008
- unter der GNU Free Documentation License.