Python Exceptions (besser) auswerten

Carsten Grohmann

8. Januar 2014 / 6. Januar 2020

Agenda

- Kurze Einführung in Python Exceptions
- Real Life
- Individueller Exception-Handler
- Ferdsch
- Weiterführende Informationen
- Nachwort

- Exceptions sind in Python das Mittel der Wahl, um mit Fehlern umgezugehen
- Exceptions sollten nach Möglichkeit nahe am Entstehungsort abgefangen und behandelt werden
- Nicht abgefangenen Exceptions
  - werden auf der Konsole angezeigt und/oder ins Log geschrieben
  - führen zu einem automatischen Programmabbruch
- Python merkt sich nur die letzte Exception (Stichwort Exception im `except:` Block)
- Exceptions enthalten den kompletten Stack inkl. lokaler und globaler Variablen ☺ (im Gegensatz zu anderen Sprachen wie Java)

```
1 #!/usr/bin/env python2
2
3 i = 1
4 j = 0
5 i/j
```

```
1 # ./donothing.py
2 Traceback (most recent call last):
3   File "./donothing.py", line 5, in <module>
4     i/j
5 ZeroDivisionError: integer division or modulo by zero
```

```
1  #!/usr/bin/env python2
2
3  import sys
4
5  i = 1
6  j = 0
7  try:
8      i/j
9  except ZeroDivisionError:
10     print "Fehler: Division durch 0"
11     sys.exit(0)
```

```
1  # ./donothing2.py
2  Fehler: Division durch 0
```

```
1  #!/usr/bin/env python2
2  f = None
3  try:
4      try:
5          content = open('/etc/motd').read()
6          foo = 'bar' in content
7      except (IOError, OSError):
8          foo = 'not found'
9  finally:
10     if f:
11         f.close()
12 print "Foo: %s" % foo
```

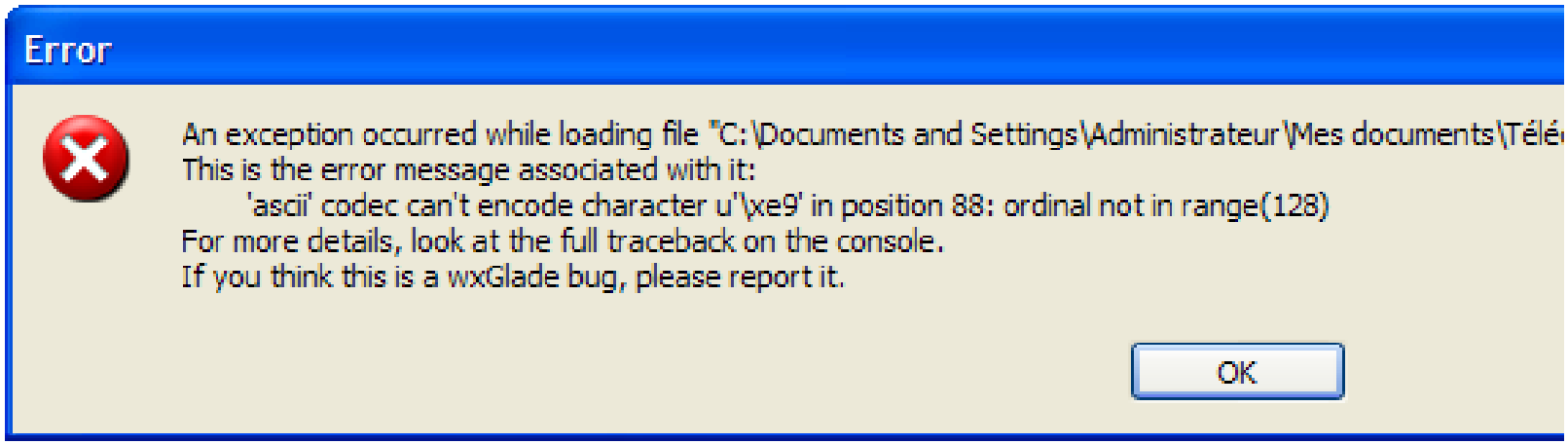
```
1  # ./catch.py
2  False
```

- Ab Py2.5 sind `try:except` und `try:finally` kombinierbar

In manchen Fehlersituationen ist es sinnvoll die Behandlung der Exception kurz auszusetzen, um zum Beispiel um vorher noch Werte zu korrigieren - Stichwort Rollback der Datenbank.

```
1  db = sqlite3.connect(MyDB)
2  try:
3      cursor = db.cursor()
4      cursor.execute('CREATE TABLE cfg (key VARCHAR(36)'
5          ' PRIMARY KEY, value TEXT NOT NULL);')
6      insert_data(cursor)
7  except:
8      db.rollback()
9      log.error('Transaction rollback')
10     raise
11 else:
12     log.info('Transaction commit')
13     db.commit()
14 return
```

- Alle möglichen Exceptions sauber abgefangen und behandelt?
- Was tun, wenn es doch passiert?
- Bekommt der Administrator / Entwickler alle notwendigen Informationen, um das Problem zu lösen bzw. zu umgehen?
- Wer schreibt perfekten Code?



Hmm, auf einem Windows mit französischen Spracheinstellungen gab es einen UnicodeEncodeError mit dem Zeichen 'u\xe9' (é - LATIN SMALL LETTER E WITH ACUTE) an Position 88.

Aber wo?!

```
1  try:
2      if not is_filelike:
3          os.chdir(os.path.dirname(infilename))
4          infile = open(infilename)
5      else:
6          infile = infilename
7      p = ProgressXmlWidgetBuilder(input_file=infile)
8      p.parse(infile)
9  except (IOError, OSError, SAXParseException), msg:
10     [...]
11     return False
12 except Exception, msg:
13     wx.MessageBox(
14         _('Exception while loading "%s"') % fn,
15     [...]
16     return False
```

Stack Trace enthält den vollständigen Inhalt des Stacks zum Zeitpunkt der Exception inklusive:

- Fehlerursache und -details
- Dateinamen und Zeilennummern
- Funktionsnamen und -code
- Lokale und globale Variablen

Individueller Exception Handler schreiben um damit den kompletten Stack Trace ins Log schreiben.

1. Alle Variablen initialisieren
2. Allgemeine Details loggen
   - 2.1 Datum und Uhrzeit
   - 2.2 Versionsnummern der Anwendung, des Python-Interpreters und wichtiger Bibliotheken
   - 2.3 Art der Exception
   - 2.4 Nachricht der Exception
3. Liste der Frames aus dem Stack Trace extrahiert (`inspect.getinnerframes()`)
4. Für jeden Frame:
   - 4.1 Kontext aus dem Quelltext loggen
   - 4.2 Lokale Variablen des Frames loggen
5. Immer alle Variablen löschen, um Zirkelbezug zu vermeiden

```
1  def exec_format(exc_type, exc_value, exc_tb):
2      <Alle Variablen initialisieren>
3      try:
4          try:
5              <Allgemeine Details loggen>
6              <Liste der Frames ermitteln>
7              for frame_details in stack_list:
8                  <Kontext loggen>
9                  <Lokale Variablen des Frames loggen>
10         except Exception, e:
11             <Aktuelle Exceptions behandeln>
12     finally:
13         <Alle Variablen loeschen>
```

Vollständige Quelltext unter MIT-Lizenz in einer erweiterten Form:

`https://hg.sr.ht/~carstengrohmann/PythonExceptionHandler`

1. Logging initialisieren
2. Exception Handler registrieren

```
1 sys.excepthook = exec_format
```

Integration in das Python-Logging ist einfach ☺:

- Eigene Klasse von `logging.Formatter` ableiten und `formatException()` entsprechend der vorangegangenen Ausführung mit Leben füllen.
- Einfache Funktion für den Exception Hook schreiben

```
1 def exec_handler(exc_type, exc_value, exc_tb):
2     logging.error(
3         "Unhandled exception occurred",
4         exc_info=(exc_type, exc_value, exc_tb)
5     )
```

- Logging mit eigenen Formatter initialisieren

```
1 default_formatter = myFormatter(
2     '%(levelname)-8s: %(message)s')
3 logger = logging.getLogger()
4 console = logging.StreamHandler()
5 console.setLevel(logging.INFO)
6 console.setFormatter(default_formatter)
7 logger.addHandler(console)
```

- Eigenen Exception Handler aktivieren

```
1 import sys
2 sys.excepthook = exec_handler
```

Ab jetzt gibt es Details von:

- nicht abgefangene Exceptions via `exec_handler()`
- abgefangenen Exceptions über den Aufruf von `logging.exception()`

```
1 # ./donothing3.py
2 ERROR    : An unhandled exception occurred
3 An unexpected error occurred!
4
5 Date and time:        2014-01-05T17:42:43.074278
6 Python version:       2.7.6
7
8 Exception type:     <type 'exceptions.ZeroDivisionError'>
9 Exception details: integer division or modulo by zero
```

```
1  Application stack trace:
2  Stack frame at level 0
3  ========================
4    File "./donothing3.py", line 8
5    Function "<module>()"
6    Source code context:
7
8        import log
9        log.init()
10
11       i = 1
12       j = 0
13   ->  i/j
```

```
1  Local variables:
2  -> i (<type 'int'>): 1
3  -> j (<type 'int'>): 0
4  -> log (<type 'module'>): <module 'log' from '/home/carste
5  -> __builtins__ (<type 'module'>): <module '__builtin__'
6  -> __file__ (<type 'str'>): './donothing3.py'
7  -> __package__ (<type 'NoneType'>): None
8  -> __name__ (<type 'str'>): '__main__'
9  -> __doc__ (<type 'NoneType'>): None
```

```
Traceback (most recent call last):
  File "./donothing.py", line 5, in <module>
    i/j
ZeroDivisionError: integer division or modulo by zero
```

vs.

```
      i = 1
      j = 0
->    i/j
  Local variables:
  -> i (<type 'int'>): 1
  -> j (<type 'int'>): 0
```

- Vollständige Quelltext

  `https://hg.sr.ht/~carstengrohmann/PythonExceptionHandler`

- The Python Tutorial - Errors and Exceptions

  `http://docs.python.org/2/tutorial/errors.html`

- Python Dokumentation des Modules „logging“

  `http://docs.python.org/2/library/logging.html`

- Python Dokumentation des Modules „inspect“

  `http://docs.python.org/2/library/inspect.html`

- Doug Hellmann - Python Exception Handling Techniques

  `http://doughellmann.com/2009/06/python-exception-handling-techniques.html`

Setzt den Exception Handler überall ein - die Lizenz erlaubt es!