



Tag 6

Exceptions

24. März 2023

Ablauf

- Rückblick Fünfter Tag
- Übung «Kommando-Interpreter» besprechen
- Exceptions (Ausnamen) Theorie
- Übung «Formelparser»

Rückblick Tag 5

Exceptions

- Weshalb 'Ausnahmen'?
- Abfangen
- Auslösen
- Vordefinierte Exceptions

Lernziele

- Exceptions-Konzept verstehen
- Exceptions fangen können
- Exceptions auslösen können
- Wissen, dass man auch eigene Exception-Klassen machen kann.
- try-(except)-finally Konstrukt lesen können

Exceptions

- Dienen der Fehlerbehandlung
- Lesen einer Datei: Datei kann ungültig sein (*open()* misslingt) oder während dem Lesen verschwinden, weil z. B. der USB-Stick aufgesteckt wird (*readline()* misslingt)

```
input_file = open('Data/2_test.txt', 'r')
while True:
    line = input_file.readline()
    if not line: break
    print(line.rstrip())
```

Alternative

- Zusätzlich zum eigentlichen Resultat (*input_file* resp. *Line*) müsste auch immer noch ein Status zurückgeliefert werden
- Dieser Status müsste jeweils auch immer geprüft werden
- Und würde ein zusätzliches Abbruchkriterium sein
 - Code wäre vermutlich 2x so lange und deutlich schwieriger zu verstehen
 - ‹produktiver› Code und Fehlerbehandlungscode vermischt

Fangen von Exceptions

try:
 <produktiver Code> |
except:
 <Fehlerbehandlung> |

```
try:  
    input_file = open('test.txt', 'r')  
    while True:  
        line = input_file.readline()  
        if not line: break  
        print(line.rstrip())
```

```
except: # catch all Exceptions  
    print(f"reading file failed")
```


Fangen von spezifischen Exceptions

`except <type>:` fängt Exceptions von einem spezifischen Type
`except ValueError:` fängt z. B. nur ValueError exceptions
`except:` fängt alle Exceptions

Reihenfolge wichtig – spezifisch zuerst, am Schluss die generellen Typen!

`except <type> as ex:`

verwenden der Exception-Informationen. z. B. um die Fehlermeldung zu erstellen

z. B.

```
except IOError as ex:  
    print(f"Error while reading file: {ex}")
```

finally

- Der finally-block wird immer, in jedem Fall durchlaufen – egal ob eine Exception geworfen wurde oder nicht. Sogar wenn ein *return* im try-Block ist.
- Perfekt geeignet, um Ressourcen freizugeben.
- Siehe auch `with .. as ..`
`with open(..) as file:`
`file.read()`

Werfen von Exceptions

- `raise <ExceptionType>(arguments)`

```
def my_function(number):  
    if not isinstance(number, int):  
        raise TypeError("'number' must be a int")  
    if number < 0:  
        raise ValueError("'number' must be positive")  
    return number + 1
```

Exception Typen

(Auswahl)

TypeError	Ungültiger Type
ValueError	Ungültiger Wert
KeyError	Schlüssel nicht vorhanden, z.B. in einem Dictionary
IndexError	Index ausserhalb des gültigen Bereiches, z.B. Liste
IOError	Ein-/Ausgabe Fehler
Exception	Mutter aller Ausnahmen

Mehr: <https://www.tutorialsteacher.com/python/error-types-in-python>

Eigene Exception Typen machen

(Vorschau OOP)

```
class MyException(Exception) :  
    pass
```

Dies erstellt eine Ableitung (Kind-Klasse) von Exception, fügt aber keine neuen Attribute (Felder oder Methoden) hinzu. Danach können diese normal geworfen und gefangen werden:

```
def foo_bar():  
    raise MyException("dont call this")  
  
try:  
    foo_bar()  
except MyException as ex:  
    print(f"The expected occurred: {ex}")
```

Übung Formelparser

- Ziel: Erstelle einen Formelparser, der Ausdrücke wie «7 + 4» parsen und berechnen kann
 - Operationen: Plus, Minus, Multiplikation, Division
 - Die Eingabe sollte aus 3 Tokens bestehen, die mit Leerzeichen getrennt sind
- Aufgaben:
 - Eingabe einlesen, in Tokens zerlegen (split)
 - Exception falls nicht 3 Tokens
 - Tokens umwandeln und Tuple in der Form (Zahl, Operator, Zahl) zurückgeben
 - Exception falls Token 1 und 3 nicht Zahlen sind
 - Tuple auswerten und Ergebnis ausgeben
 - Exception falls Operator nicht in `[+, -, *, /]` ist
 - Das ganze in einer Endlos-Schleife mit Fangen und Anzeigen von Exceptions
- Erweiterungen:
 - Spezifische *ParserException* Klasse
 - Zahl soll nicht nur *float* sein, sondern *int* falls die Eingabe ganzzahlig ist
 - Ausgabe der Datentypen
 - Unit test für die Funktionen