Merkblatt - Exceptions

May 7, 2020

1 Exceptions

Bei der Ausführung eines Programms kann es passieren, dass ein Fehler auftritt.

Beispielsweise kann dies bei einer Division durch 0 passieren, oder auch, wenn du versuchst, auf eine Datei zuzugreifen, die nicht (mehr) existiert:

[1]:
```
print(5 / 0)
```

```
␣
↪---------------------------------------------------------------------------

ZeroDivisionError                         Traceback (most recent call␣
↪last)

<ipython-input-1-ca8321cec4b1> in <module>()
----> 1 print(5 / 0)


ZeroDivisionError: division by zero
```

[2]:
```
with open("datei.xyz", "r") as file:
    print(file)
```

```
␣
↪---------------------------------------------------------------------------

FileNotFoundError                         Traceback (most recent call␣
↪last)

<ipython-input-2-44c2cd2a495d> in <module>()
----> 1 with open("datei.xyz", "r") as file:
      2     print(file)
```

```
FileNotFoundError: [Errno 2] No such file or directory: 'datei.xyz'
```

Manchmal möchtest du nicht, dass bei einem Fehler das Programm direkt beendet wird.

Mit einem `try … except` - Block kannst du diese Fehler abfangen, und darauf reagieren:

[3]:
```
try:
    print(5 / 0)
    print(4)
except ZeroDivisionError:
    print("Durch null teilen ist nicht erlaubt!")
print(5)
```

```
Durch null teilen ist nicht erlaubt!
5
```

1.1 Mehrere `try … except` - Blöcke

Dein Programm kann auch mehrere Fehler per `except` abfangen und darauf reagieren:

[3]:
```
try:
    with open("datei.xyz", "r") as file:
        print(file)
    print(5 / 0)
except ZeroDivisionError:
    print("Du darfst nicht durch 0 teilen")
except FileNotFoundError:
    print("FileNotFoundError ist aufgetreten")
```

```
FileNotFoundError ist aufgetreten
```

1.2 Eigene Fehler auslösen

Über den `raise` - Befehl kannst du eigene Fehler auslösen:

[8]:
```
class InvalidEmailError(Exception):
    pass

def send_mail(email, subject, content):
    if not "@" in email:
        raise InvalidEmailError("email does not contain an @")
try:
    send_mail("hallo", "Betreff", "Inhalt")
except InvalidEmailError:
    print("Bitte gebe eine gültige E-Mail ein")
```

```
Bitte gebe eine gültige E-Mail ein
```

1.3 Mit `finally` aufräumen

Wenn du möchtest, dass ein bestimmter Codeblock auf jeden Fall ausgeführt wird, egal, ob ein Fehler auftritt oder nicht, kannst du diesen Code in einen `finally` - Block schreiben. Dieser Code wird auf jeden Fall ausgeführt, selbst wenn ein Fehler vorher aufgetreten ist.

In dem Fall hier z. B. kannst du dadurch garantieren, dass du eine einmal geöffnete Datei auf jeden Fall über das `.close()` schließt (notwendig, wenn du die Datei nicht über ein `with file = open("existiert.txt", "r")` öffnest).

Andere Beispiele könnten z. B. sein, dass eine Netzwerkverbindung auf jeden Fall noch getrennt wird, etc.

[9]:
```
try:
    file = open("existiert.txt", "r")
    print(file)
    print(5 / 0)
except FileNotFoundError:
    print("Datei wurde nicht gefunden")
finally:
    print("FINALLY!!!")
    file.close()
```

```
<_io.TextIOWrapper name='existiert.txt' mode='r' encoding='UTF-8'>
FINALLY!!!


      ␣
↪---------------------------------------------------------------------------

       ZeroDivisionError                         Traceback (most recent call␣
↪last)

       <ipython-input-9-6e4494755c61> in <module>()
         2     file = open("existiert.txt", "r")
         3     print(file)
   ----> 4     print(5 / 0)
         5 except FileNotFoundError:
         6     print("Datei wurde nicht gefunden")


       ZeroDivisionError: division by zero
```

1.3.1 Das `with` - Konstrukt

In der Praxis bietet sich aber für Dateien primär das `with` - Konstrukt an. Da ist quasi schon seitens Python implementiert, dass die Datei auf jeden Fall geschlossen wird - egal, ob ein Fehler auftritt oder nicht.

Unser Code wird also sehr viel übersichtlicher:

[6]:
```
with open("existiert.txt", "r") as file:
    print(file)
```

```
<_io.TextIOWrapper name='existiert.txt' mode='r' encoding='UTF-8'>
```

[ ]:
```

```