Würgeschlange 3 - Lesson 6

Tobias Maschek, Viktor Reusch
https://github.com/jemx/wise1920-python

mit Materialien von Felix Döring, Felix Wittwer https://github.com/fsr/python-lessons

3. Dezember 2019

Python-Kurs

Gliederung

1. I/O

2. Exceptions

I/O

Datei einlesen

```
# hello_world.txt:
#    Hello
#    World
#    !
with open("hello_world.txt") as file:
    for line in file:
        print(line)
# Output:
#    Hello
#
#    World
#
#    !
#
```

with

`with` sorgt dafür, dass die Datei nach dem Ende des Einlese-Blocks wieder geschlossen wird.

Datei schreiben

```
with open("numbers.txt", mode="w") as file:
    for i in range(3):
        file.write(f"{i}\n")
# numbers.txt:
#   0
#   1
#   2
```

mode

Es gibt verschidene Modi, mit denen man eine Datei öffnen kann:

r	einlesen *(Standard)*
w	schreiben
w+	lesen und schreiben
a	an bestehende Datei anfügen

Aufgabe 6-1

Lese und schreibe eine Datei:

- Lese die Zeilen einer Text-Datei in eine Liste.
- Drehe die Reihenfolge der Elemente in der Liste um.
- Schreibe die umgedrehte Liste der Zeilen in eine neue Datei.

Exceptions

Always except an exception!

Viele Funktionen in Python können *exceptions* werfen, so auch open.

```
1 with open("this.file.does.not.exist") as file:
2     pass
```

Oh S***! *Wir wollen ja auch nicht demonetarisiert werden.*

```
1 Traceback (most recent call last):
2   File "<stdin>", line 1, in <module>
3 FileNotFoundError: [Errno 2] No such file or directory: 'this.file.does.not.
      exist'
```

Handling Exceptions

```
try:
    open("this.file.does.not.exist")
except FileNotFoundError:
    print("File not found!")
# The file does not exist, Python couldn't find it!
```

try In diesem Block wird eine potentiell fehlschlagende Funktion **versucht** auszuführen.

except Sollte ein Fehler auftreten, „fängt“ dieser Block den Fehler auf.

Bei einem Fehler wird der try-Block direkt abgebrochen und der Rest des Codes **nicht** mehr ausgeführt. Entsteht kein Fehler, wird der except-Block nicht ausgeführt.

Fehler über Fehler!

```
try:
    open("formula.for.cold.fusion")
except FileNotFoundError:
    print("File not found!")
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    open("formula.for.cold.fusion")
PermissionError: [Errno 13] Permission denied: 'formula.for.cold.fusion'
```

Da kein except-Block für PermissionError existiert, fangen wir diesen Fehler nicht auf.

Lösungsmöglichkeiten

```python
try:
    open("formula.for.cold.fusion")
except FileNotFoundError:
    print("File not found!")
except PermissionError as e: # All exceptions inherit from Exception
    print(f"I have no permission!\nSee the error message: {e}")
# I have no permission!
# See the error message: [Errno 13] Permission denied: (...)
```

```python
try:
    open("formula.for.cold.fusion")
except (FileNotFoundError, PermissionError):
    print("Could not read file!")
```

```python
try:
    open("formula.for.cold.fusion")
except Exception as e:
    print(e)
# [Errno 13] Permission denied: 'formula.for.cold.fusion'
```

Fehler-Schlacht!1!11!!elf!

raise wird ähnlich wie return benutz, wirft aber den gegeben Fehler.
Entweder wird der Fehler hoffentlich durch ein try aufgefangen oder das Programm schmiert ab.

Eigene *Exceptions/Errors* erben von *Exception*.

```
1 class CustomError(Exception):
2     def __init__(self, number):
3         super().__init__(f"The number must be 0 or 1 not {number}")
4 def fail_proof_function(number):
5     if not isinstance(number, int):
6         raise TypeError("number is not an int")
7     if number != 0 and number != 1:
8         raise CustomError(number)
9 fail_proof_function(2) # __main__.CustomError: The number must be 0 or 1 not 2
```

Finally, we speak about finally!

Der **finally**-Block wird in jedem Fall ausgeführt. Also nach dem try- oder except-Block.

```
1 file = None
2 try:
3     file = open("a.file")
4 except Exception as e:
5     print(e)
6 finally:
7     if file is not None:
8         file.close()
```

with

Öffnet man eine Datei unter Nutzung eines **with**-Blocks, wird beim Verlassen des Blockes garantiert geschlossen. also auch, wenn ein Fehler fliegt.

Aufgabe 6-2

Schreibe ein Python-Programm, dass

- die *input()* Funktion verwendet, um zwei Zahlen einzulesen,
- diese beiden Zahlen dividiert und ausgibt (als *int*).
- Sollte der Benutzer etwas anderes als einen *int* eingeben, soll eine entsprechende Fehlermeldung ausgegeben werden.