Programmierung mit Python

Grundlagen Grundbegriffe Grundübungen

Alfatraining Kurs 2026-06 Dozent: Karsten Flügge

Offene Themen / Spezialsyntax

Fehlerbehandlung try except
@dekaratoren

Iteratoren + yield eigenes for i in iterator
immutable types str tupel

@dataclass
@classmethod

@dekarator objekte?

… Vertiefungen

… spezielle module (interne, externe) / Anwendungen

z.B. random, *numpy* (numerische Berechnungen, Matrizen und Vectoren),

GEMEINSAME INTERESSEN:
SQL, AI?, GUI, **json, github/gitlab**

Fehlerbehebung

In Python werden Fehler oft mit einer (Kontroll)**Struktur** namens **`try-except`** behandelt. Das ermöglicht es, auf Fehler zu reagieren und das Programm trotzdem fortzusetzen.

- **try-Block**: Dieser Block enthält Code, von dem man erwartet, dass er möglicherweise einen Fehler verursachen könnte. Python führt diesen Code zunächst normal aus.
- **except-Block**: Wenn im try-Block ein Fehler auftritt, springt Python in den except-Block, wo der Fehler behandelt werden kann. Wenn kein Fehler auftritt, wird der except-Block übersprungen.

```
try:
  x = 1 / 0  # Versuch, durch Null zu teilen, was einen Fehler verursacht
except: # unbekannte Fehlerart 'auffangen' (catch)
  print("Es wurde versucht, durch Null zu teilen.")
```

Fehlerbehebung

```
try:
  x = 1 / 0  # Versuch, durch Null zu teilen, was einen Fehler
verursacht
  print("kein Fehler aufgetreten")
except: # unbekannte Fehlerart 'auffangen' (catch)
  print("Es wurde versucht, durch Null zu teilen.")
else:
  print("wenn kein Fehler aufgetreten ist.") # ähnlich kein break
finally:
  print("Dieser Block wird immer ausgeführt")

print("Dieser Block wird auch immer ausgeführt")
```

Fehlerbehebung

```
def test():
    try:
        return 42
    finally:
        print("wird immer aufgerufen trotz return")

print(test())
print("danach")
```

Fehlerbehebung

sicherstellen, dass Verbindung sauber geschlossen wird

```
def finally_a_use_case():
 verbindung = False
 try:
  verbindung = open_connection()
  verbindung.send(data)
 finally:
  if verbindung:
   verbindung.close()
```

Fehler Objekt

- Statt nur des Types kann man auch ein konkretes **Fehler Objekt** fangen:

```
try:
  dangerous_function()
except Exception as e: # e ist neue Variable die unseren Fehler enthält
  print(f"Dieser Fehler ist aufgetreten: {e}")
```

die Benennung der lokalen Fehlervariable e ist frei (innerhalb der Benamungsregeln).

"Exception" ist generischer Fehler, andere **erben** davon

```
try:
  1 / 0
except ZeroDivisionError as e: # bekannte Fehlerart
  print(f"Dieser Fehler ist aufgetreten: {e}")
```

Gute Praxis:

Wenn möglich **Ausnahmen als Fälle** behandeln, nicht mit try: except:

```
try:
  y/x
except ZeroDivisionError:
  print("Es wurde versucht, durch Null zu teilen.")


if x == 0:
 return math.inf # 1/0 = infinity
 # print("kann nicht durch 0 teilen")
else:
 return y/x
```

Gute Praxis II:

Wenn möglich **die Fehlerbehandlung einfach halten,** so dass nicht neue Fehler erzeugt werden

```
try:
  dangerous_function()
except:
  print("Unbekannte Fehlerart")
  try_to_fix_situation() # möglichst keine Fehler werfen!!
```

Nicht so schön:

```
try:
  dangerous_function()
except:
  try:
    try_to_fix_situation()
  except:
    print("Fehler bei Fehlerbehandlung")
```

Fehler Nichtbehebung

- Man kann Fehler bei Bedarf ignorieren

```
try:
  x = 1 / 0  # Versuch, durch Null zu teilen, was einen Fehler verursacht
except:
  pass # passe, ignoriere
```

Fehlerbehebung in main

- Es ist üblich praxis, im Hauptprogramm einfach alle Fehler zu '**fangen**'

```
try:
  mein_programm()
except: # catch
   aufräumen() # Dateien schliessen, löschen
  print("irgendwas ging schief")
```

```
if __name__ == '__main__': # heisst: Datei direkt gestartet, nicht importiert
  main()
```

Kleine Aufgabe

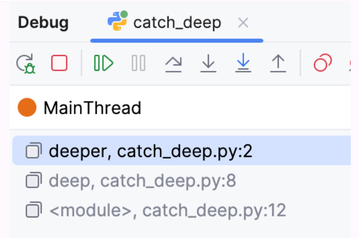
Fange 1/0

Stack trace

Beim Erzeugen von Fehlern wird automatisch eine Aufrufs**hierarchie** erzeugt:
Daran erkennt man wie wir zu dem Problem gekommen sind.

traceback / callstack

```
Traceback (most recent call last):
  File "/Users/me/Documents/ALFA/alfatraining_python_3/code/alfa_python/
tag11a/catch_deep.py", line 12, in <module>
    deep()
    ~~~~^^
  File "/Users/me/Documents/ALFA/alfatraining_python_3/code/alfa_python/
tag11a/catch_deep.py", line 8, in deep
    deeper()
    ~~~~~~^^
  File "/Users/me/Documents/ALFA/alfatraining_python_3/code/alfa_python/
tag11a/catch_deep.py", line 3, in deeper
    raise Exception("too deep")
Exception: too deep
```



Fehler Hierarchie

```
|  Method resolution order:
  |      ZeroDivisionError
  |      ArithmeticError
  |      Exception
  |      BaseException
  |      object
```

es wird der erste 'passende' Fehler aufgefangen,
daher müssen spezifische Unterklassen zuerst gefangen werden.

Es wird stets nur der erste *passende* except block behandelt.
Passend heisst, dass der geworfene Fehlertyp erbt,
z.b. ZeroDivisionError < ArithmeticError

Fehler Melden

- Man kann selber Fehler melden '**werfen**' mit dem **raise keyword**

- **mit "raise" teilt man Python mit, dass ein Fehler aufgetreten ist --> bewusste Fehlersituation**

```
try:
   if system_state_dirty():
     raise Exception("hier ist etwas faul!")  # throw in anderen sprachen
  komplizierte_andere_funktionen() # wird dann nicht erreicht!!
except Exception as e:
  print(f"Dieser Fehler ist aufgetreten: {e}")
```

raise ≈ heben (Hand heben;)

Folgecode wird übersprungen, wir landen direkt in Fehlerbahandlungsblock 'except'

Eigene Fehler Typen

```
class MyException(Exception):
    pass


try:
  raise MyException("MEin Fehler ist aufgetreten")
except MyException as e:
  print(e)
except Exception as e:
  print(f"Unbekannter Fehler: {e}")
```

Fehler Weitermelden

Wenn man bestimmte Fälle nicht an Ort und Stelle beheben kann: **Problem *eskalieren*.**

```
try:
  if system_state_dirty():
    raise Exception("hier ist etwas faul!")  # throw in anderen sprachen
except Exception as e:
 print(f"Dieser Fehler ist aufgetreten: {e} Was soll ich tun?")
  raise  # Fehler weiter reichen 'rethrow'
```

Fehler ≈ Ausnahmen

- Die Oberklasse von allen Fehlern ist "**Exception**"
- Ausnahmen müssen keine "Fehler" sein, sondern z.b. **Sonderfälle**
- **z.B. KeyboardInterruption,** StopIteration
- Man kann auch davon erben um Eigene Ausnahmen zu behandeln

```
class MeinFehler(Exception):
  pass
```

```
try:
  raise MeinFehler("Ein Fehler ist aufgetreten")
except Exception as e:
# except MeinFehler as e:
  print(e)
```

Eigene Ausnahmen

- Die Oberklasse von allen Fehlern ist "**BaseException**"
- Man kann auch davon erben um Eigene Ausnahmen zu behandeln

```
class MeinFehler(Exception):
  pass


try:
  raise MeinFehler("Ein Fehler ist aufgetreten")
except MeinFehler as e:
  print(e)
```

Fehlerbehebung verschiedener Typen (Error/Exception)

```python
try:
  dangerous_function()
  # Exception ≈ Error
except ZeroDivisionError:
  print("Es wurde versucht, durch Null zu teilen.")
except Exception as e:
  print(f"Ein Fehler ist aufgetreten: {e}")
except:
  print(f"IRGENDEIN Fehler ist aufgetreten: {e}")
```

catch all

- Wenn man nicht sicher ist, was für Fehler ein Block '**werfen**' kann, so kann man beliebige Fehler '**fangen**', indem man except ohne Typ schreibt:

```
try:
  dangerous_function()
except:
   try_to_fix_situation()
   print("Unbekannter Fehler")
```

Traceback

Traceback / stacktrace / backtrace
ist der Pfad welcher zu dem Fehler geführt hat

```
/opt/homebrew/bin/python /Users/me/alfa_python/tag9/raise.py
Traceback (most recent call last):
  File "/Users/me/alfa_python/tag9/raise.py", line 14, in <module>
    can_handle_error()
  File "/Users/me/alfa_python/tag9/raise.py", line 11, in can_handle_error
    raise e
  File "/Users/me/alfa_python/tag9/raise.py", line 7, in can_handle_error
    raise Exception("hier ist etwas faul!")  # throw in anderen sprachen
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

^^^^ genau die Stelle wo der Fehler auftrat
```

Verwand mit dem call stack

Aufgaben

```
class MeinFehler(Exception): # eigene Fehler Klasse für 2) (Vorschau für Montag)
  pass
```

- 1. Erzeuge programmatisch (ohne raise) einen Fehler und behandle ihn
- 2. **Was für Fehler sind uns schon über den Weg gelaufen**? Alle verursachen + fangen!
- 3. **Erzeuge einen eigenen Fehler und zeige Fehlerdetails**
  - (Tipp: Syntax war except Exception as e:)
- 4. Erstelle zwei eigene Fehler Klassen und 'werfe' ein Objekt davon, fange beide Typen
- 4b. was passiert wenn man in 3 einen 'catch-all' except Exception: voranstellt?

5. Zusatzaufgabe (extra/schwer) : schreibe ein kleines Beispiel Programm welches mit try: except: mehrmals versucht eine Aktion durchzuführen "repair/reentry pattern"

Aufgaben Erkenntnis

- **Wenn eine Oberklasse vor einer Unterklasse behandelt wird,**
- **wird der Fall der Unterklasse nie erreicht!!**
- **CATCH ALL immer am Ende**

Fehlerbehandlung immer vom spezifischen zum allgemeinen Fall

```
try:
   1/0
except ZeroDivisionError:
   print("Es wurde versucht, durch Null zu teilen.") # wird nie erreicht!!
except Exception:
   print(f"Ein Fehler ist aufgetreten")
except:
   print(e)
```

Aufgaben Erkenntnis

- **Bekannte Fehler:**

- **my_dict["unknown_key"] KeyError**
- **my_dict[unknown_key] NameError**
- **my_list[index_too_big] IndexError**
- **my_list.xyz AttributeError: 'str' object has no attribute**
- **3 + "vier" TypeError**
- **a # NameError: name 'a' is not defined**
- **my_str[0]='A' TypeError 'str' object does not support item assignment**
  - **import xyz ModuleNotFoundError**
  - **open("does_not_exist.txt") FileNotFoundError**
  - **i=int('Hallo') ValueError**
  - **def loop():loop() RecursionError**

Kompilerfehler lassen sich NICHT mit try: except: abfangen

- **SyntaxError**
- **IntentationError**

Pause

FRÜHSTÜCK bis 10:20

Python for scripting / modding

Games and applications using python as **scripting language:**

Blender - Allows Python scripting for creating mods, automating tasks, and extending the software's capabilities. **AutoCAD oder**
Autodesk Maya - Supports Python for writing plugins and scripts that modify and extend functionality, often used in professional animation and modeling environments.
GIMP - Uses Python for developing plugins and extensions that enhance or modify the image editing capabilities of the software.
Microsoft XLS - soon

VIELE VIELE MEHR

Civilization IV - Allows extensive modding with Python, enabling modders to tweak game mechanics, units, and AI behavior.
World of Tanks - Uses Python for modding, particularly for customizing user interfaces and gameplay elements.
The Sims 4 - Supports Python for creating mods that alter gameplay, though this is done through the game's API rather than direct scripting support.
Eve Online - Python is used to develop mods and scripts that enhance the game's interface and functionalities.
Minecraft? - https://education.minecraft.net/en-us/resources/computer-science-subject-kit/python-101

Unreal Engine 20% **aller Spiele?!? - https://docs.unrealengine.com/5.4/en-US/PythonAPI/**

Pause

```
def f(a,b,c):
 pass

~5

f(1,2,3) # positionell
f(a=1,c=2,b=3) # keyword / name parameter

literal Werte:
int: 1,2,3  float: 1.2 string: "adf"

right binding
a**b**c und a=b=c=4
x = a if c1 else b if c2 else d
# = a if c1 else (b if c2 else d)
```

Pause

Keywords

False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	
return				
and	continue	for	lambda	try
as	def	from	*nonlocal*	while
assert	del	*global*	not	with
async	elif	if	or	yield
match	case			