Ausnahmenbehandlung

Fehlerbehandlung

- Ausnahmen objektorientiert
- hier: erster Teil
- keine eigenen Ausnahmen

Fehlerbehandlung

- brauchen Namen des Fehlers
- einmal auslösen

```
>>> 5 + "vier"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +:
'int' and 'str'
```

Fehlerbehandlung

Funktion add() kann

in Zeile 2 TypeError

auslösen

```
1 def add ( x, y ):
2     summe = x + y
3     return summe
```

Fehlerbehandlung

try … except

erlaubt

Fehlerbehandlung

```
1  def add ( x, y ):
2      try:
3          summe = x + y
4
5      except TypeError:
6          print("Falsche Operanden für Summe")
7          return None
8
9      return summe
```

Fehlerbehandlung

```python
def add ( x, y ):
    try:
        summe = x + y

    except TypeError:
        print("Falsche Operanden für Summe")
        return None

    return summe

print("Summe: ", add( "fünf", 4 ) )
```

```
Falsche Operanden für Summe
Summe:  None
```

Fehlerbehandlung

Fehler abfangen

- komplettes try...except

```
try:
    ...
except TypeError as e:
    ...
except:
    ...
else:
    ...
finally:
    ...
```

Fehlerbehandlung

mit as bekommen

wir Variable für Fehler

args[0] enthält

Details

```
1  def add ( x, y ):
2      try:
3          summe = x + y
4
5      except TypeError as e:
6          print( e.args[0] )
7          return None
8
9      return summe
10
11 print("Summe: ", add( "fünf", 4 ) )

can only concatenate str (not "int") to str
Summe:  None
```

Fehlerbehandlung

mehrere Fehler als

Tupel

```
1  def add ( x, y ):
2      try:
3          summe = x + y
4
5      except (TypeError, IndexError) as e:
6          print( e.args[0] )
7          return None
8
9      return summe
10
11 print("Summe: ", add( "fünf", 4 ) )

can only concatenate str (not "int") to str
Summe:  None
```

Fehlerbehandlung

Fehler abfangen -- mehrere

- in separaten except-Zweigen

```
1 def add(x,y):
2     try:
3         x+y
4     except IndexError as e:
5         print("Index ",e.args[0])
6     except TypeError as e:
7         print("Typ ",e.args[0])
```

Fehlerbehandlung

Fehler abfangen -- mehrere

- except alleine:

  alle anderen

- nur an letzter

  Stelle

```
1 def add(x,y):
2     try:
3         x+y
4     except IndexError as e:
5         print("Index ",e.args[0])
6     except TypeError as e:
7         print("Typ ",e.args[0])
8     except:
9         print("Irgendetwas anderes.")
```

Fehlerbehandlung

Fehler abfangen -- else

- else-Zweig

  ausgeführt wenn

  kein Fehler

```
1  def add(x,y):
2      try:
3          z = x+y
4      except TypeError as e:
5          print("Typ ",e.args[0])
6      else:
7          print("Hier else.")
8      return z
9
10 print(add(1,2))

Hier else.
3
```

Fehlerbehandlung

Fehler abfangen -- finally

- finally-Zweig immer ausgeführt

```
1  def add(x,y):
2      try:
3          z = x+y
4      except TypeError as e:
5          print("Typ ",e.args[0])
6      else:
7          print("Hier else.")
8      finally:
9          print("Hier finally.")
10     return z
11 print(add(1,2))

Hier else.
Hier finally.
3
```

Fehlerbehandlung

Fehler selbst auslösen

raise

```
1 def add ( x, y ):
2     if not isinstance(x, int):
3         raise TypeError
4     summe = x + y
5     return summe
6
7 print("Summe: ", add( "fünf", 4 ) )
```

Fehlerbehandlung

Fehler selbst auslösen

raise

mit Beschreibung

```
1 def add ( x, y ):
2     if not isinstance(x, int):
3         raise TypeError("keine Strings")
4     summe = x + y
5     return summe
6
7 print("Summe: ", add( "fünf", 4 ) )


TypeError: keine Strings
```

Fehlerbehandlung

Fehler werden "weitergereicht"

raise ohne Argument:

gibt letzten behandelten

Fehler weiter an eine

Ebene höher

```
1  def attic(x):
2      assert x != 0
3      return 1 / x
4  def floor(x):
5      try:
6          attic(x)
7      except:   raise
8  try:
9      x = floor(0)
10 except RuntimeError:
11     x = -3
12 except:
13     x = -2
14 else:
15     x = -1
16 print(x)
```

Fehlerbehandlung

Zusicherungen

assert legt Muss-Bedingung fest

wenn nicht erfüllt: AssertionError

Fehlerbehandlung

Zusicherungen

assert

```
>>> lst = [7, 1, 3, 5, -12]

>>> assert max(lst) == 7

>>> assert min(lst) == -12

>>> assert sum(lst) == 0
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
AssertionError
```

What is the expected outcome of the following code?

```
try:
    raise IndexError

except TypeError:
    print("a")
except IndexError:
    print("b")
except:
    print("c")
```

a) b

b) a

c) Syntax Error

d) c