

# Grundlagen der Programmierung

*Vorlesung und Übung*

## 08 – Exceptions und Assertions

Prof. Dr. Andreas Biesdorf

Wirtschaft  
Hauptcampus

H O C H  
S C H U L E  
T R I E R

- Was passiert, wenn eine Funktion in einen **unerwarteten Zustand** kommt?
- Es wird eine **Ausnahme** („Exception“) ausgelöst:

## TYPISCHE EXCEPTIONS – EINFACH ZU KORRIGIEREN

- **Zugriffe jenseits der Größe von Listen**  
`test = [1,2,3]`  
`test[4]` → `IndexError`
- **Konvertierungen von inkompatiblen Typen**  
`int(test)` → `TypeError`
- **Referenzierung von nicht existierenden Variablen**  
`a` → `NameError`
- **Vermischung von Datentypen**  
`'3'/4` → `TypeError`
- **Vergessen von Klammern**  
`a= len([1,2,3]`  
`print a` → `SyntaxError`

## Weitere häufige Fehlertypen:

- `SyntaxError`: Programm kann nicht interpretiert werden
- `NameError`: lokaler oder globaler Variablenname fehlt
- `AttributeError`: Referenz auf Attribut schlägt fehl
- `TypeError`: Operand hat den falschen Typ
- `ValueError`: Wert ist illegal
- `IOError`: Zugriff auf Dateisystem schlägt fehl

## „Fail Silently“

- Ersetze Werte mit Defaults und mache weiter
- Risiko: Nutzer erhält keine Warnung, dass ein Fehler aufgetreten ist

## Gib dem Nutzer einen Fehlerwert zurück

- Erhöht die Komplexität im Code

## Beende Programmausführung

- Sende Signal an Nutzer / Entwickler
- `raise Exception („Beschreibung der Ausnahme“)`

## Allgemein:

```
try:
    a = int(input("Tell me one number:"))
    b = int(input("Tell me another number:"))
    print(a/b)
    print ("Okay")
except:
    print("Bug in user input.")
print("Outside")
```

## Speziell:

```
try:
    a = int(input("Tell me one number: "))
    b = int(input("Tell me another number: "))
    print("a/b = ", a/b)
    print("a+b = ", a+b)
except ValueError:
    print("Could not convert to a number.")
except ZeroDivisionError:
    print("Can't divide by zero")
except:
    print("Something went very wrong.")
```

### else

- Wenn der try-Body ohne Fehler ausgeführt wurde

### finally

- Immer ausgeführt nach try, else und except, selbst wenn break/continue/return ausgeführt wurde
- Nützlich, um den Code am Ende aufzuräumen (z.B. Datei schließen)

# WELCHE EXCEPTION WIRD GEWORFEN?

- `'1' / '2 '`
- `'1' / 2`
- `mylist = [10, 20, 30]`  
`mylist.index(11)`
- `A=2`  
`3*a`

?

SyntaxError

ValueError

TypeError

NameError

## Idee:

Wiederholen der Eingabe solange, bis der korrekte Datentyp verwendet wurde

```
while True:
    try:
        n = input("Please enter an integer: ")
        n = int(n)
        break
    except ValueError:
        print("Input not an integer – try again")
print("Correct input of an integer!")
```

## Achtung:

Funktioniert nur für `ValueError`



## Idee: Öffnen von Dateien

```
data = []

file_name = input("Provide a name of a file of data ")

try:
    fh = open(file_name, 'r')
except IOError:
    print('cannot open', file_name)
else:
    for new in fh:
        if new != '\n':
            addIt = new[:-1].split(',') # remove trailing \n
            data.append(addIt)

    fh.close() # close file even if fail
```

- **Vermeidung der Rückgabe von „Spezialwerten“ bei Fehlern**
- Alternative: Werfen einer **Exception**

```
raise <Exception-Typ> (<Argumente>)
```

```
raise ValueError ("Fehler aufgetreten!")
```

Schlüsselwort

Typ der Exception

Argumente

```
def get_ratios(L1, L2):  
    """ Assumes: L1 and L2 are lists of equal length of numbers  
    Returns: a list containing L1[i]/L2[i] """  
    ratios = []  
    for index in range(len(L1)):  
        try:  
            ratios.append(L1[index]/float(L2[index]))  
        except ZeroDivisionError:  
            ratios.append(float('NaN')) # NaN = Not a Number  
        except:  
            raise ValueError('get_ratios called with bad arg')  
    return ratios
```

- 1) Handling spezieller Exceptions
- 2) Werfen eigener Exception

Welche Ausgabe erzeugt die folgende Funktion bei den gegebenen Eingaben?  
Nicht behandelte Fehler als „Error“ benennen

```
def fancy_divide(numbers, index):  
    try:  
        denom = numbers[index]  
        for i in range(len(numbers)):  
            numbers[i] /= denom  
    except IndexError:  
        print("-1")  
    else:  
        print("1")  
    finally:  
        print("0")
```

```
fancy_divide([0, 2, 4], 1)
```

```
fancy_divide([0, 2, 4], 4)
```

```
fancy_divide([0, 2, 4], 0)
```

Welche Ausgabe erzeugt die folgende Funktion bei den gegebenen Eingaben?  
Nicht behandelte Fehler als „Error“ benennen

```
def fancy_divide(numbers, index):  
    try:  
        denom = numbers[index]  
        for i in range(len(numbers)):  
            numbers[i] /= denom  
    except IndexError:  
        fancy_divide(numbers, len(numbers) - 1)  
    except ZeroDivisionError:  
        print("-2")  
    else:  
        print("1")  
    finally:  
        print("0")
```

```
fancy_divide([0, 2, 4], 1)
```

```
fancy_divide([0, 2, 4], 4)
```

```
fancy_divide([0, 2, 4], 0)
```

Welche Ausgabe erzeugt die folgende Funktion bei den gegebenen Eingaben?  
Nicht behandelte Fehler als „Error“ benennen

```
def fancy_divide(numbers, index):  
    try:  
        try:  
            denom = numbers[index]  
            for i in range(len(numbers)):  
                numbers[i] /= denom  
        except IndexError:  
            fancy_divide(numbers, len(numbers) - 1)  
        else:  
            print("1")  
        finally:  
            print("0")  
    except ZeroDivisionError:  
        print("-2")
```

fancy\_divide([0, 2, 4], 1)

fancy\_divide([0, 2, 4], 4)

fancy\_divide([0, 2, 4], 0)

Welche Ausgabe erzeugt die folgende Funktion bei den gegebenen Eingaben?  
Nicht behandelte Fehler als „Error“ benennen

```
def fancy_divide(list_of_numbers, index):  
    try:  
        try:  
            raise Exception("0")  
        finally:  
            denom = list_of_numbers[index]  
            for i in range(len(list_of_numbers)):  
                list_of_numbers[i] /= denom  
    except Exception as ex:  
        print(ex)
```

```
fancy_divide([0, 2, 4], 0)
```

Welche Ausgabe erzeugt die folgende Funktion bei den gegebenen Eingaben?  
Nicht behandelte Fehler als „Error“ benennen

```
def fancy_divide(list_of_numbers, index):  
    try:  
        try:  
            denom = list_of_numbers[index]  
            for i in range(len(list_of_numbers)):  
                list_of_numbers[i] /= denom  
        finally:  
            raise Exception("0")  
    except Exception as ex:  
        print(ex)
```

```
fancy_divide([0, 2, 4], 0)
```



`fancy_divide([0, 2, 4], 0)` erzeugt einen `ZeroDivisionError`. Bitte korrigieren Sie die Funktion `simple_divide`! Bei Division mit 0 soll eine Liste mit dem Wert „0“ zurückgegeben werden.

```
def fancy_divide(list_of_numbers, index):  
    denom = list_of_numbers[index]  
    return [simple_divide(item, denom) for item in list_of_numbers]  
  
def simple_divide(item, denom):  
    return item / denom
```

- Weiteres Stilmittel des “Defensiven Programmierens”
- Sicherstellung, dass der **Vertrag** beim Aufruf einer Funktion eingehalten wird
- Eine Violation des Assert-Statements wirft eine `AssertionError` Exception

```
def avg(grades):  
    assert not len(grades) == 0, 'no grades data'  
    return sum(grades)/len(grades)
```

Sofortige Terminierung der Funktion, wenn die Bedingung nicht eingehalten wurde