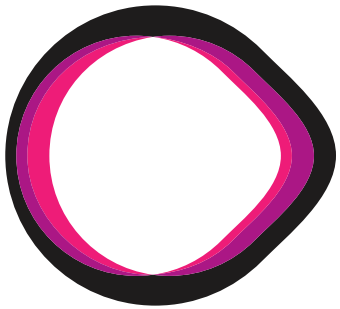
Python – Lektion 4
Tupel Packing, Funktionsparameter und Comprehensions



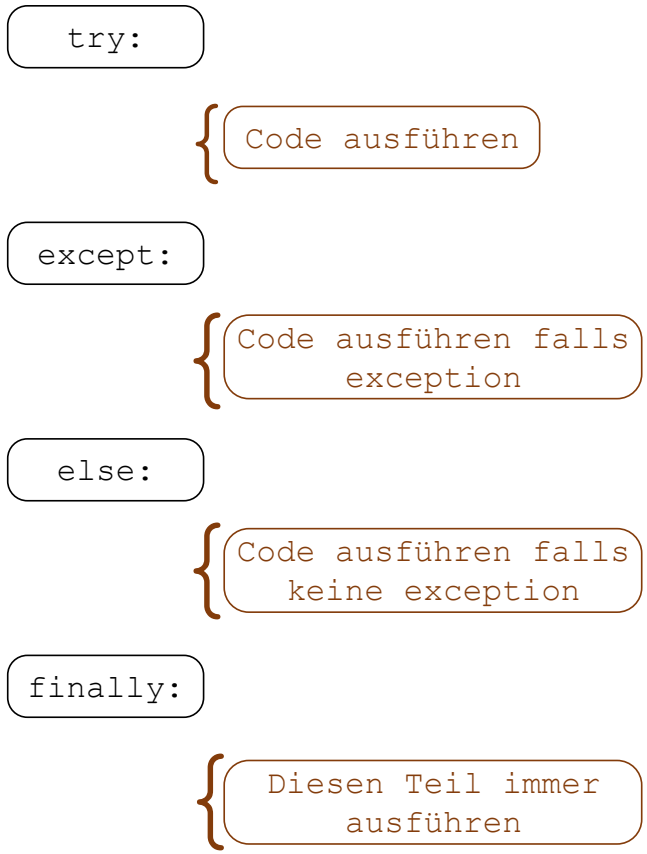
OST
Ostschweizer
Fachhochschule

Disclaimer

Die Vertraulichkeitsklasse dieser Daten ist ”intern-erweitert”. Sie dürfen die Daten als OST-Angehörige nutzen, aber nicht an Dritte weitergeben oder veröffentlichen.

Ausnahmebehandlung

`try`, `except`, `finally`, `raise`



Rückblick

Stringformatierung

- Beispiel `format()`

```
"Die Summe aus {0:.2f} und {1} ist {summe:>6.2f}".format(zahl1,
                                                         zahl2,
                                                         summe=summe1)
```

- Beispiel `f-string`

```
f"Die Summe aus {zahl1:.2f} und {zahl2} ist {zahl1 + zahl2:>6.2f}"
```

- Die eigentliche Formatierung:
  `{[<name>][!<conversion>][:<format_spec>]}`
- `[:<format_spec>]` im Detail:
  `:[<fill>][<align>][<sign>][#][0][<width>][<group>][.<precision>][<type>]`

Alles über Strings - Stringmethoden

- `split`, `join`, `replace`, `strip`, `lower`, ...

Heutige Themen

- Tupel Packing und Unpacking
- Funktionsparameter im Detail
- Abstraktionen / Comprehensions

Tupel Packing und Unpacking

`http://localhost:8888/notebooks/tupel.ipynb`

Funktionen - Funktionsparameter Rückblick

```
def summe(a, b, c=0, d=0):
    return a + b + c + d
```

- Aufruf mit Positionsargumenten (*Positional Arguments*):

```
>>> summe(1, 2)
3
>>> summe(1, 2, 3, 4)
10
```

- Aufruf mit Schlüsselwortargumenten (*Keyword Arguments*):

```
>>> summe(d=4, c=3, b=2, a=1)
10
```

- Kombination aus beiden (Positional *vor* Keyword Arguments):

```
>>> summe(1, 2, d=4, c=3)
10
```

Funktionen - Funktionsparameter im Detail

```
>>> max(10, 50)
50

>>> max(10, 50, 60)
60
```

Funktionsaufruf mit beliebiger Anzahl von Argumenten:

`http://localhost:8888/notebooks/funktionen.ipynb`

* Operatoren Zusammenfassung

`foo(1, 2, 3, d=4, e=5)` → `def foo(a, *args, **kwargs):`

- `*` packt überschüssige Positionsargumente in ein Tupel
- `**` packt überschüssige Schlüsselwortargumente in ein Dictionary

* Operatoren Zusammenfassung

`foo(*noten)` → `foo(4.5, 6, 4)`

Iterable mit `*` entpacken und als Positionsargumente übergeben

`foo(**noten)` → `foo(ET=4.5, Py=6, Comm=4)`

Dictionary mit `**` entpacken und als Schlüsselwortargumente übergeben

* Operatoren Zusammenfassung

`[*noten, 4, 5, 6]` → `[4.5, 6, 4, 4, 5, 6]`

Iterable mit * in eine Liste entpacken

`{**noten, **noten2}` → `{"ET": 4.5, "Py": 6, "Comm": 4, "Sig": 5}`

Dictionaries mit ** in anderes Dictionary entpacken

Abstraktionen / Comprehensions

- Comprehensions

  Erzeugungsvorschrift für die Erzeugung von Instanzen iterierbarer Objekte wie:

  - Listen
  - Dictionaries
  - Sets

`http://localhost:8888/notebooks/comprehension.ipynb`

Vorteile Comprehensions:

- Einfach bestehende Listen, Dicts, etc abändern/filtern
- Alles auf einer Zeile $\rightarrow$ übersichtlich

List Comprehension

Konventionell (mit oder ohne `if` bzw. mit oder ohne `if`/`else`):

```
values = list()
for item in iterable:
     if condition:
        values.append(expression1)
     else:
        values.append(expression2)
```

mit `if` am Ende für das Filtern von Elementen:

```
values = [expression1
          for item in iterable
          if condition]
```

mit inline `if` vor der for-Schleife für die Entscheidung zwischen zwei möglichen Ausdrücken[1]:

```
values = [expression1
          if condition else expression2
          for item in iterable]
```

[1] verschachtelte `if/else` sind auch möglich

Set Comprehension

Konventionell (mit oder ohne `if` bzw. mit oder ohne `if`/`else`):

```
values = set()
for item in iterable:
     if condition:
        values.add(expression1)
     else:
        values.add(expression2)
```

mit `if` am Ende für das Filtern von Elementen:

```
values = {expression1
          for item in iterable
          if condition}
```

mit inline `if` vor der for-Schleife für die Entscheidung zwischen zwei möglichen Ausdrücken[2]:

```
values = {expression1
          if condition else expression2
          for item in iterable}
```

[2] verschachtelte `if/else` sind auch möglich

Dict Comprehension

Konventionell (mit oder ohne `if` bzw. mit oder ohne `if`/`else`):

```
values = dict()
for item in iterable:
     if condition:
        values[item] = expression1
     else:
        values[item] = expression2
```

mit `if` am Ende für das Filtern von Elementen:

```
values = {item:expression1
          for item in iterable
          if condition}
```

mit inline `if` vor der for-Schleife für die Entscheidung zwischen zwei möglichen Ausdrücken[3]:

```
values = {item:expression1
          if condition else item:expression2
          for item in iterable}
```

[3] verschachtelte `if/else` sind auch möglich