# Vertiefung der Python Konzepte

| Parameter | Kursinformationen |
| --- | --- |
| Veranstaltung: | `Prozedurale Programmierung / Einführung in die Informatik / Erhebung, Analyse und Visualisierung digitaler Daten` |
| Semester | `Wintersemester 2025/26` |
| Hochschule: | `Technische Universität Freiberg` |
| Inhalte: | `Erweiterte Konzepte der Programmiersprache Python` |
| Link auf Repository: | https://github.com/TUBAF-IfI-LiaScript/VL_EAVD/blob/master/09_PythonVertiefung.md |
| Autoren | Sebastian Zug & André Dietrich & Galina Rudolf |


**Fragen an die heutige Veranstaltung ...**

- Welche Standarddatentypen exisiteren in Python über die Liste hinaus?
- In welchen Anwendungsfällen kommen diese zum Einsatz?
- Wie lassen sich Funktionen mit Python realsieren und welche Unterschiede existieren im Vergleich zu C++?

## Weitere Datentypen

Sie haben bereits Listen (`list`), `range` Objekte und Text (`string`) als Datenstruktur kennengelernt - im Weiteren existieren daneben vier weitere Sequenzdatentypen: byte sequences (`bytes` objects), byte arrays (`bytearray` objects) und `tuples`. Dazu kommen `dictionaries` und `sets` als Containertypen.

| Datentyp | Besonderheit | Syntax |
|---|---|---|
| `list` | veränderliche Sequenz von (beliebigen) Daten | `l = ["grün", 1, True]` |
| | | `l[0] = 4` |
| `strings` | Darstellung von Zeichenketten | `s="Hello World"` |
| `bytes` | Unveränderbare Folge von Elementen | `t1 = (1, 2, 3)` |
| `bytearray` | Unveränderbare Folge von Elementen | `t1 = (1, 2, 3)` |
| `tupel` | Unveränderbare Folge von Elementen | `t1 = (1, 2, 3)` |
| `range` | iterierbare, unveränderbare Folge von Elementen | `r = range(3, 20, 2)` |

# Tupel oder Liste?

Warum ein Tupel anstelle einer Liste verwenden?

- Die Programmausführung ist beim Iterieren eines Tupels schneller als bei der entsprechenden Liste - Dies wird wahrscheinlich nicht auffallen, wenn die Liste oder das Tupel klein ist.
- Der Speicherbedarf eines Tupels fällt in der Regel geringer aus, als bei einer Listendarstellung ein und des selben Inhaltes.
- Manchmal sollen Daten unveränderlich gehalten werden - Tupel schützen die Informationen vor versehentlichen Änderungen.

**size.py**

```python
import sys

a_list = []
a_tuple = ()
a_list = ["Hello", "TU", "Freiberg"]
a_tuple = ("Hello", "TU", "Freiberg")
print("List data size : " + str(sys.getsizeof(a_list)))
print("Tupel data size: " + str(sys.getsizeof(a_tuple)))
```

```
List data size : 88
Tupel data size: 64
List data size : 88
Tupel data size: 64
```

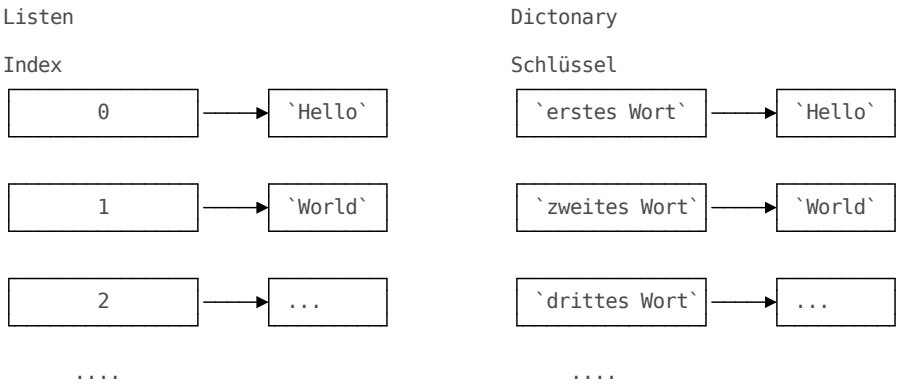
**duration.py**

```python
import time

l=list(range(10000001))
t=tuple(range(10000001))

start = time.time_ns()
for i in range(len(t)):
    a = t[i]
end = time.time_ns()
print("Tuple duration: ", end - start)

start = time.time_ns()
for i in range(len(l)):
    a = l[i]
end = time.time_ns()
print("List duration : ", end - start)
```

## Dictionaries

Dictionaries werden zur Speicherung von Schlüssel Werte Paaren genutzt. Ein dictionary ist eine Sammlung von geordneten (entsprechend der Reihenfolge der "Einlagerung"), veränderlichen Einträgen, für die Schlüssel werden keine Dublikate zugelassen.

Ein Telefonbuch ist das traditionelle Beispiel für eine Implementierung des Dictonaries. Anhand der Namen werden die Telefonnummern zugeordnet.




**dictonary.py**

```
1  capital_city = {"France": "Paris",
2                  "Italy": "Rome",
3                  "England": "London"}
4
5  print(capital_city)
6  print("England" in capital_city)
7
8  capital_city["Germany"] = "Berlin"  # add a single value
9
10 examples = {"Belgium": "Bruessels",
11                "Poland": "Warsaw"}
12
13 capital_city.update(examples)        # add a single or multiple new en
14                                      # contained in a dictionary
15 print(capital_city)
16
```

```
{'France': 'Paris', 'Italy': 'Rome', 'England': 'London'}
True
{'France': 'Paris', 'Italy': 'Rome', 'England': 'London', 'Germany':
'Berlin', 'Belgium': 'Bruessels', 'Poland': 'Warsaw'}
{'France': 'Paris', 'Italy': 'Rome', 'England': 'London'}
True
{'France': 'Paris', 'Italy': 'Rome', 'England': 'London', 'Germany':
'Berlin', 'Belgium': 'Bruessels', 'Poland': 'Warsaw'}
```

**wrongdictonary.py**

```
oldtimer =  {"brand": "VW",
             "model": "Käfer",
             "year": 1964,
             "year": 2020
}
print(oldtimer)
```

```
{'brand': 'VW', 'model': 'Käfer', 'year': 2020}
```

Nehmen wir an, Sie entwerfen ein Verzeichnis der Studierenden aus Freiberg. Sie wollen die Paarung Studierendenname zu Matrikel als Dictonary umsetzen. Einer Ihrer Kommilitonen schlägt vor, dafür zwei Listen zu verwenden und die Verknüpfung über den Index zu realisieren. Was meinen Sie dazu?

**goodSolution.py**

```
students = {"von Cotta": 12,
            "Humboldt": 17,
            "Zeuner": 233}

student = "von Cotta"
print(f"Student  {student} ({students[student]})")
```

```
Student  von Cotta (12)
```

**badSolution.py**

```
names = ["von Cotta", "Humboldt", "Zeuner"]
matrikel = [12, 17, 233]

i = 1
print(f"Student  {names[i]} ({matrikel[i]})")
```

```
Student  Humboldt (17)
Student  Humboldt (17)
```

## Sets

Ein Set ist eine Sammlung, die ungeordnet, unveränderlich (in Bezug auf exisiterende Einträge) und nicht indiziert ist. Kernelement der Idee ist das Verbot von Dublikaten.

**set.py**

```
fruits = {"apple", "banana", "cherry", "apple", "apple"}

print(fruits)
```

```
{'apple', 'banana', 'cherry'}
{'apple', 'banana', 'cherry'}
```

Die Leistungsfähigkeit von Sets resultiert aus den zugehörigen Mengenoperationen.

**set.py**

```
a = {1,2,3,4,5,6}
b = {2,4,6,7,8,9}
even = {2,4,6,8,}

print(8 in a)
print(even < b)    # ist even eine Teilmenge von b?
print(a | b)       # Vereinigung von a und b
print(a & b)       # Schnittmenge von a und b
print(b - a)       # welche Einträge existieren in b die nicht in a prä
  sind
```

```
False
True
{1, 2, 3, 4, 5, 6, 7, 8, 9}
{2, 4, 6}
{8, 9, 7}
```

## Zusammenfassung

**DataTypeExample.py**

```python
for i in ['a','b','c']:      # Liste
  print(i, end=",")
print()

for i in "abc":              # String
  print(i, end=",")
print()

for i in ('a','b','c'):      # Tupel
  print(i, end=",")
print()

for i in {0:"a", 1:"b"}:     # Dictonary
  print(i, end=",")
print()

for i in {'a','b','c','c'}: # Set
  print(i, end=",")
```

```
a,b,c,
a,b,c,
a,b,c,
0,1,
b,c,a,a,b,c,
a,b,c,
a,b,c,
0,1,
a,b,c,
```

Gegeben sei eine Liste der Studiengangsbezeichnungen für die Studierenden dieser Vorlesung. Leiten Sie aus der Liste ab

1. Wie viele Studierende eingeschrieben sind?
2. Wie viele Studiengänge in der Veranstaltung präsent sind?
3. Wie viele Studierende zu den Studiengängen gehören?

```python
# Angabe der Studiengänge der 2022 eingeschriebenen Teilnehmer in der
# Veranstaltung
topics = [
  "S-UWE", "S-WIW", "S-GÖ", "S-VT", "S-GÖ", "S-BAF", "S-VT",
  "S-WWT", "S-NT", "S-WIW", "S-ET", "S-WWT", "S-MB", "S-WIW",
  "S-FWK", "F1-INF", "S-WIW", "S-BWL", "S-WIW", "S-MAG",
  "F2-ANCH", "S-MAG", "S-WWT", "S-NT", "S-ACW", "S-GTB",
  "S-WIW", "F2-ANCH", "S-GTB", "S-GÖ", "S-GBG", "S-GM",
```

```
  "S-MAG", "S-GTB", "S-WIW", "S-WIW", "S-FWK", "S-WIW",
  "S-MAG", "S-GBG", "S-GÖ", "S-BAF", "S-BAF", "S-NT", "S-GÖ",
  "S-WWT", "S-GBG", "S-WWT", "S-GBG", "S-ERW", "S-WWT",
  "S-WIW", "S-NT", "S-WIW", "S-GÖ", "S-WIW", "S-GM",
  "S-GBG", "F1-INF", "S-WIW", "S-WWT", "S-ACW", "S-WIW",
  "S-WWT", "S-ACW", "S-INA", "S-FWK", "S-GTB", "S-WIW",
  "S-MORE", "S-WIW", "S-GÖ", "S-BWL", "S-CH", "S-WIW",
  "F2-ANCH", "S-WIW", "S-ACW", "S-ET", "S-ET", "S-GÖ",
  "S-GÖ"
]

# zu 1
print(len(topics))                          # Länge der Liste

# zu 2
print(len(set(topics)))                     # Anzahl individueller Ei
                                            # als Größe des Sets

# zu 3
print({i:topics.count(i) for i in topics})  # Auftretenshäufigkeit al
                                            # "List" Comprehension
                                            # Dictonary der Einträge
```

```
82
22
{'S-UWE': 1, 'S-WIW': 18, 'S-GÖ': 9, 'S-VT': 2, 'S-BAF': 3, 'S-WWT': 8,
'S-NT': 4, 'S-ET': 3, 'S-MB': 1, 'S-FWK': 3, 'F1-INF': 2, 'S-BWL': 2,
'S-MAG': 4, 'F2-ANCH': 3, 'S-ACW': 4, 'S-GTB': 4, 'S-GBG': 5, 'S-GM':
2, 'S-ERW': 1, 'S-INA': 1, 'S-MORE': 1, 'S-CH': 1}
```

## Eigene Funktionen

> Das kennen wir schon ... aber noch mal zur Sicherheit

*Funktionen sind Unterprogramme, die ein Ausgangsproblem in kleine, möglicherweise wiederverwendbare Codeelemente zerlegen.*

**Bessere Lesbarkeit**

Der Quellcode eines Programms kann schnell mehrere tausend Zeilen umfassen. Beim Linux Kernel sind es sogar über 15 Millionen Zeilen und Windows, das ebenfalls zum Großteil in C geschrieben wurde, umfasst schätzungsweise auch mehrere Millionen Zeilen. Um dennoch die Lesbarkeit des Programms zu gewährleisten, ist die Modularisierung unerlässlich.

**Wiederverwendbarkeit**

In fast jedem Programm tauchen die gleichen Problemstellungen mehrmals auf. Oft gilt dies auch für unterschiedliche Applikationen. Da nur Parameter und Rückgabetyp für die Benutzung einer Funktion bekannt sein müssen, erleichtert dies die Wiederverwendbarkeit. Um die Implementierungsdetails muss sich der Entwickler dann nicht mehr kümmern.

**Wartbarkeit**

Fehler lassen sich durch die Modularisierung leichter finden und beheben. Darüber hinaus ist es leichter, weitere Funktionalitäten hinzuzufügen oder zu ändern.

In allen 3 Aspekten ist der Vorteil in der Kapselung der Funktionalität zu suchen.

## Syntax

> Funktionsdefinition starten immer mit dem Schlüsselwort **def**. Typen für Parameter oder Rückgabewerte müssen nicht angegeben werden! Es gelten die üblichen Einrückungsregeln.

```
def funktionsname(arg1, arg2, ...):
  <anweisungen>
```

**FunctionExample.py**

```
1  import math
2
3  def print_pi():
4    print(math.pi)
```

## Parameterübergabe

```
1  def halbiere(zahl):
2    zahl = zahl / 2
3    print(zahl)
4
5  zahl = 5
6  print( halbiere(zahl) )
```

```
2.5
None
2.5
None
```

Python erlaubt analog zu C++ die Vergabe von Standardparametern beim Funktionsaufruf.

## Returnwerte

Mit **return** kann **ein** Rückgabewert festgelegt und die Funktion beendet werden. Der Rückgabewert kann auch ein Tupel, eine Liste oder ein beliebiges anderes Objekt sein. Tupel können später wieder in einzelne Variablen augetrennt werden.

```
1  def halbiere(zahl):
2    zahl = zahl / 2
3    return zahl
4
5  zahl = 5
6  print(halbiere(zahl))
7  print(zahl)
```

Der Parameter `zahl` wird als *Call-by-Assignment* übergeben. Die Zuweisung eines neuen Wertes ändert nicht die gleichnamige Variable außerhalb der Funktion!

`return` erlaubt lediglich einen Rückgabewert. Wie handhaben wir dann die Situation, wenn es mehrere sind?

```
1  def sumAndMultiplyTo(n):
2    sum = 0
3    prod = 1
4    for i in range(1,n+1):
5      sum = sum + i
6      prod = prod * i
7    return sum,prod
8
9  result = sumAndMultiplyTo(10)
10 print( type(result) )
11 print(result)
12
13 x,y = sumAndMultiplyTo(10)
14 print(x)
15 print(y)
```

```
<class 'tuple'>
(55, 3628800)
55
3628800
```

## Typ-Hinweise für Variablen

**Variante 1: Der Interpreter meckert ...**

```python
from typing import List

def my_function(numbers: List[int]) -> int:
    return sum(numbers)

numbers = [1, 2, 3, 4, 5]
print(my_function(numbers))
```

```
15
15
```

**Variante 2: Die Entwicklungsumgebung meckert ...**

```python
def sum_number(a: int, b: int) -> int:
    return a+b

sum_number(1, 1.01)
```

```
Argument of type "float" cannot be assigned to parameter "b" of type "int" in function "sum_number"
  "float" is incompatible with "int" Pylance(reportGeneralTypeIssues)
View Problem (Alt+F8)  Quick Fix... (Ctrl+.)
```

## Beispiel der Woche

Wir nutzen das Newton-Verfahren zur näherungsweisen Berechnung einer Nullstelle. Gesucht wird die Quadratwurzel einer Zahl *a*.

Funktion $f(x) = x^2 - a$, so dass für die Nullstelle gilt: $x^2 = a$

1. In jedem Schritt berechnen wir $x_{n+1} = x_n - f(x_n)/f'(x_n)$.
2. Für unseren Fall: $x_{n+1} = \frac{1}{2}(x_n + \frac{a}{x_n})$
3. Wir beenden die Iteration, wenn $|x_{n+1} - x_n| < \varepsilon$

**newton.py**

```
1  def square_root(start, a, output=False, eps=0.00000001):
2    xn = a
3    while True:
4      if output: print(xn)
5
6      # x = xn - (xn**2 - a) / (2*xn)
7      # oder:
8      x = (xn + a/xn) / 2
9
10     if abs(x-xn) < eps:
11       break
12     xn = x
13
14   return x
15
16 if __name__ == "__main__":
17   x = float( input("Inital value for x:") )
18   output = input("Show all outputs (y/n)?")
19   if output == 'y':
20     output = True
21   else:
22     output = False
23
24   a=4
25   result = square_root(x, a, output)
26   print(f"Zero point of x^2-{a}", result)
27
```

```
Inital value for x:Inital value for x:
```

Welches Problem sehen Sie?

# Quiz

## Weitere Datentypen

# Tupel oder Liste?

Welche Vorteile hat der Datentyp `Tupel` gegenüber dem Datentyp `Liste`?

- [ ] `Tupel` sind einfacher zu erstellen.
- [ ] Iterationen über `Tupel` sind schneller als Iterationen über `Listen`
- [ ] `Tupel` können mehr Elemente enthalten
- [ ] `Tupel` benötigen weniger Speicherbedarf

Wie lautet die Ausgabe dieses Programms?

```
a = (9, 2)
print(a[0])
```

- ◯ 9
- ◯ 2
- ◯ Das Programm endet mit einem Error

Wie lautet die Ausgabe dieses Programms?

```
a = (9, 2)
a[0] = 7
print(a[0])
```

- ◯ 9
- ◯ 7
- ◯ 2
- ◯ Das Programm endet mit einem Error

## Dictionaries

Wie lautet die Ausgabe dieses Programms?

```python
# Hier werden Noten gespeichert
grades = {"Peter": 1.0,
          "Franz": 3.0,
          "Max": 1.7,
          "Jonas": 2.3}

examples = {"Kurt": 1.3,
            "Bernd": 3.3}

examples["Michi"] = 2.0

grades.update(examples)
print(grades["Michi"])
```

- ◯ 2.0
- ◯ Das Programm endet mit einem Error

## Sets

Wie viele Elemente befinden sich im Set `st`?

```python
st = {"Franz", "Peter", "Franz", "Michi", "Peter"}
print(st)
```

Wie lautet die Ausgabe dieser Funktion? (Bitte geben Sie die Antwort ohne geschweifte Klammern an)

```python
a = {1,2,3,4}
b = {7,4,6,7}

print(b - a)
```

Wie lautet die Ausgabe dieser Funktion? (Bitte geben Sie die Antwort ohne geschweifte Klammern an)

```
a = {1,2,3,4}
b = {7,4,6,7}

print(b & a)
```

Wie lautet die Ausgabe dieser Funktion? (Bitte geben Sie die Antwort ohne geschweifte Klammern an)

```
a = {1,2,3,4}
b = {7,4,6,7}

print(b < a)
```

- True
- False

Wie lautet die Ausgabe dieser Funktion? (Bitte geben Sie die Antwort ohne geschweifte Klammern oder Leerzeichen an)

```
a = {1,2,3,4}
b = {7,4,6,7}

print(b | a)
```

## Eigene Funktionen

# Syntax

Mit welchem Schlüsselwort starten Funktionsdefinitionen in Python?

## Parameterübergabe

Wie lautet die Ausgabe dieses Programms auf 2 Nachkommastellen gerundet?

```
from math import pi

def to_rad(num):
  rad = num * (pi / 180)
  return rad

deg = 90
print(to_rad(deg))
```

## Returnwerte

Wie lautet die Ausgabe dieses Programms? Bitte geben Sie die Antwort ohne Klammern an.

```
def get_min_max(a):
    return (min(a), max(a))

a = (10, 47, 18, 1, 33, 20)
result = get_min_max(a)
print(result)
```

Wie lautet die Ausgabe dieses Programms?

```
def modify_number(a):
```

```
def modify_number(a):
    a = -1
    return

a = 42
modify_number(a)
print(a)
```