Matthias Mruzek-Vering, M.Sc.

Hochschule Karlsruhe
University of
Applied Sciences

DSCB 230 - Aufgabenblatt 2

Die Aufgabenblätter werden in dieser Veranstaltung in Jupyter Notebooks veröffentlicht und bearbeitet. Diese finden Sie in der Github Organisation für Data Science 2 unter dem Repository *dscb230-tutorial* (https://github.com/hka-mmv/dscb230-tutorial). Die Musterlösung wird ebenfalls in Form eines Jupyter Notebook in Github hochgeladen.

Aufgabenteil 1

Aufgabe A: Wiederholung - List und Dict in Python3

1. Eine Liste erstellen und ein Element hinzufügen
2. Ein Element an der zweiten Stelle hinzufügen
3. Ein Element entfernen
4. Erstellen Sie eine zweite Liste und konkatenieren Sie diese
5. Zerschneiden Sie die neue Liste an zwei beliebigen Stellen
6. Erstellen Sie ein beliebiges Dictionary
7. Fügen Sie dem Dict ein weiters Element hinzu
8. Entfernen Sie aus dem Dict das neue Element.
9. Geben Sie mittels einer For-Schleife, alle Keys und Values aus dem Dict aus.

Aufgabe B: Funktionales Programmieren mittels Filter, Map und Reduce

Schreiben Sie eine Python Programm, auf Basis der Funktionalen Programmierung.

1. Erstellen Sie zunächst eine Liste mit Zufallszahlen zwischen 10 und 10.000. Die Länge sollte positiv sein, aber nicht über 10 liegen. Nun soll nach Ausführung nur noch die geraden Zahlen in einer Liste vorhanden sein.
2. Schreiben Sie nun eine Funktion welche überprüft ob eine übergebene Nummer gerade ist. Die Funktion sollte nur aus einer Zeile bestehen (Tipp: Modulo Operator)
3. Rufen Sie ihre Funktion nach dem Prinzip der Funktionalen Programmierung auf, extrahieren Sie alle geraden Zahlen und speichern Sie diese in einer Liste.

Aufgabe C: Shallow Copy und Deep Copy in Python3

Schreiben Sie ein Programm Ihrer Wahl, es sollte Shallow Copy und Deep Copy realisieren. Erklären Sie anschließend einem der 3 Tutoren Ihr Programm und was der Unterschied zwischen Shallow- und Deep Copy ist.

Aufgabe D: Nur für die Fortgeschritten!

Implementieren Sie Mergesort in Python3.

Tim Ehmann
Fabian Juny
Raphaele Salvatore Licciardo, B.Sc.

Aufgabenteil 2

1. Bäckerei

Idee aus: Stephenson, B.: The Python Workbook. A Brief Introduction with Exercises and Solutions. Texts in Computer Science. 2nd ed. 2019. Springer International Publishing; Imprint: Springer, Cham 2019, S.22

Eine Bäckerei speichert alle ihre Waren über ein Dictionary ab. Zu jedem Produkt wird die **Menge** und der **Stückpreis** (in €) angegeben. Es wird zudem gespeichert, ob die Ware **vom Vortag** ist, oder nicht.

```
waren = [

{„name": "brezel", "menge" : 17, "preis" : 0.85, „vom_vortag" :
False},

{"name": "laugenstange", "menge" : 4, "preis" : 0.90,
"vom_vortag" : True},

{"name": "mohnbrötchen", "menge" : 9, "preis" : 0.78,
"vom_vortag" : True},

{"name": "vollkornbrot", "menge" : 3, "preis" : 2.16,
"vom_vortag" : False},

{"name": "crossiant", "menge" : 12, "preis" : 0.93,
"vom_vortag" : False},

{"name": "sesambrötchen", "menge" : 13, "preis" : 0.76,
"vom_vortag" : True}]
```

1.1 Bestimmen Sie das Produkt mit dem höchsten Preis. Geben Sie den Namen und den Preis an.

1.2 Auf Produkte vom Vortag gibt es 15% Rabatt. Ändern Sie den Preis aller Produkte vom Vortag auf den neuen Angebotspreis (in „waren")

1.3 Nutzen Sie die filter() Funktion, um alle Produkte vom Vortag in einer Liste zu speichern (Name und Preis).

1.4 Angenommen die Bäckerei möchte Inventur durchführen. Dazu soll eine Kopie des kompletten Warenbestands gemacht werden. Änderungen im (aktuellen) Warenbestand oder der Inventur sollen keinen Einfluss aufeinander haben. (Überprüfen Sie danach ihre Listen, indem Sie einzelne Werte verändern und vergleichen)

2. Benotungssystem

Ein Professor möchte beim Korrigieren seiner Klausuren Zeit sparen und hat sich folgendes Benotungssystem ausgedacht:

- Beginnt der Vorname eines Student mit einem "F", so bekommt er eine 1.0
- Andernfalls bekommt der Student eine zufällige Note zwischen 2 und 5 zugeordnet

Nutzen Sie die **map()** Funktion, um das System zu implementieren.
In einer Liste soll für jeden Student ein Tuple mit Name und Note gespeichert werden.

Testen Sie ihren Code mit dem Tuple „studenten".

```
studenten = ("Peter", "Julia", "Bob", "Fabian", "Tim", "Fiola")
```

3. CSV

Schreiben Sie eine Funktion, die die Anzahl von Zeilen einer CSV Datei zurückgibt

Aufgabenteil 3

Aufgabe 1: Basics Wiederholung

1. Fügen sie die zwei Listen **treibstoffe** und **preise** zu einer Liste **treibstoffpreise** aus Tupeln mithilfe von **zip()** zusammen

2. Aktualisieren Sie für Super E10 den Preis zu 1.87 in **treibstoffpreise**

3. Entfernen Sie den Eintrag für CNG aus **treibstoffpreise**

4. Geben Sie **dieselpreise** ab dem dritten Eintrag aus

5. Geben Sie **dieselpreise** im Monat Februar aus (Index 3 bis 6)

6. Geben Sie jeden zweiten Eintrag von **dieselpreise** aus (start bei Index 0)

```
treibstoff = ["Diesel", "LKW Diesel", "Super E10", "Super Plus",
"LPG", "CNG"]

preise = [2.08, 2.09, 1.94, 2.08, 1.10, 1.09]


dieselpreise = [("2022-01-17", 1.408), ("2022-01-24", 1.418),
("2022-01-31", 1.430), ("2022-02-07", 1.441),
("2022-02-14", 1.457), ("2022-02-21", 1.464),
("2022-02-28", 1.520), ("2022-03-07", 1.719),
(„2022-03-14", 2.023), ("2022-03-21", 1.828)]
```

Matthias Mruzek-Vering, M.Sc.

Aufgabe 2:

Im letzten Tutorium wurde ein Dictionary **freundschaften** erstellt, in dem für jeden User die Namen der Freunde gespeichert werden. Zudem wurde eine Funktion **anzahl_an_freunden** erstellt, welche einen User als Input nimmt und dann mithilfe des **freundschaften** dictionaries die Anzahl an Freunden ausgibt.

Versuchen Sie nun, eine Liste so zu erstellen, in der für jeden Nutzer der Name und die Anzahl dessen Freunde als Tupel gespeichert wird. z.B. [('Paul', 2), ('Marie', 3), ...]

```
nutzer = [
    {"id": 0, "name": "Paul"},
    {"id": 1, "name": "Marie"},
    {"id": 2, "name": "Sue"},
    {"id": 3, "name": "Justus"},
    {"id": 4, "name": "Anna"},
    {"id": 5, "name": "Marian"},
    {"id": 6, "name": "Svenja"},
    {"id": 7, "name": "Devin"},
    {"id": 8, "name": "Lars"},
    {"id": 9, "name": "Peter"},
]

freundschaften = {
    'Paul': ['Marie', 'Sue'],
    'Marie': ['Paul', 'Sue', 'Justus'],
    'Sue': ['Paul', 'Marie', 'Justus', 'Lars'],
    'Justus': ['Marie', 'Sue', 'Anna'],
    'Anna': ['Justus', 'Marian'],
    'Marian': ['Anna', 'Svenja', 'Devin'],
    'Svenja': ['Marian', 'Lars'],
    'Devin': ['Marian', 'Lars'],
    'Lars': ['Sue', 'Svenja', 'Devin', 'Peter'],
    'Peter': ['Lars']}

def anzahl_an_freunden(nutzer):
    """Gibt Die Anzahl an Freunden eines Nutzers aus"""
    name = nutzer["name"]
    return len(freundschaften[name])

freundesanzahl_nach_name = [

]
```

Tim Ehmann
Fabian Juny
Raphaele Salvatore Licciardo, B.Sc.

Aufgabe 3:

Sortieren Sie nun die Liste absteigend nach Der Anzahl der Freunde. Verwenden Sie dafür die .sort() Methode.
Falls die vorherige Aufgabe nicht bearbeitet wurde, verwenden Sie die bereitgestellte Liste.

```
"""
freundesanzahl_nach_name = [('Paul', 2), ('Marie', 3), ('Sue',
4), ('Justus', 3),
('Anna', 2), ('Marian', 3), ('Svenja', 2), ('Devin', 2), ('Lars',
4), ('Peter', 1)]
"""
```

Hilfreiche Ressourcen:

https://pythonguides.com/python-sort-list-of-tuples/

Zur Verständnis Vertiefung:

https://stackoverflow.com/questions/8966538/syntax-behind-sortedkey-lambda#answer-42966511