Matthias Mruzek-Vering, M.Sc.

Hochschule Karlsruhe
University of
Applied Sciences

DSCB 230 - Aufgabenblatt 1

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: Algorithmen und Python3 Wiederholung

Implementieren Sie den Quicksort Algorithmus in Python3. Dafür haben Sie bereits einige Strukturen vorgegeben (siehe Jupyter Notebook).

```
def quicksort(arr):
    if len(arr) <= 1:
        return arr

    left =
    middle =
    right =

print(quicksort([3, 6, 8, 10, 1, 2, 1]))
```

Aufgabe B: 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.

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

Aufgabe C: Implementieren Sie einen Warenkorb

Dieser sollte über die Kommandozeile gesteuert werden, Einkaufswünsche entgegen nehmen und in den Warenkorb legen, ist der Einkauf abgeschlossen, wird die Gesamtsumme und der Warenkorb angezeigt. Hierfür haben Sie bereits die Artikelliste.

```
articles = {
    1: {
        "name": "T-Shirt",
        "price": 9.99,
        "currency": "EUR",
        "available": "67"
    },
    1: {
        "name": "Pullover",
        "price": 19.99,
        "currency": "EUR",
        "available": "19"
    },
    1: {
        "name": "Watch",
        "price": 109.99,
        "currency": "EUR",
        "available": "0"
    }
}
```

Aufgabenteil 2

Aufgabe A:

Schreiben Sie ein Programm, welches die Quersumme einer zufällig generierten Integer Zahl zwischen 0 und 100000 ausgibt.

```
import random
```

Aufgabe B:

In einer social media App können verschiedene User sich als Freunde adden. Gegeben haben wir eine Liste an Nutzern (Nutzer) und Freundschaften (*freundschaftspaare*). Um leichter sehen zu können, welche Personen am Populärsten sind bzw. welche Personen viele oder wenige Freunde auf der App haben, sollen Sie ein Dictionary *Freundschaften* erstellen, welches für jeden User die Namen der Freunde speichert. Dieses sollte so aussehen: {"Paul": ["Marie", "Sue"], ... }

Tipp1: Lege zuerst ein Dictionary an, welches nur die Namen als Keys und eine leere Liste als Value hat. Befülle die Listen erst im nächsten Schritt.

Tipp2: Die ID des Nutzers ist gleichzeitig auch der Index des Nutzers in der Nutzer Liste

```
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"},
]

freundschaftspaare = [(0, 1), (0, 2), (1, 2), (1, 3), (2, 3), (2,
8), (3, 4), (4, 5), (5, 6), (5, 7), (6, 8), (7, 8), (8, 9)]s
```

Aufgabe C:

Schreiben Sie eine Funktion *anzahl_an_Freunden*, welche ausgibt, wie viele Freunde eine Person hat. Falls die vorherige Aufgabe nicht gelöst wurde, verwenden Sie das bereitgestellte Dictionary.

```
"""
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 aus"""
    pass
```

Probieren Sie nun die Funktionalität Ihres Codes.

```
print(f"Marie hat {anzahl_an_freunden(nutzer[1])} Freunde")
print(f"Justus hat {anzahl_an_freunden(nutzer[3])} Freunde")
print(f"Peter hat {anzahl_an_freunden(nutzer[9])} Freunde")
```

Aufgabenteil 3

Aufgabe A: Schaltjahr

Schreiben Sie ein Programm, das bestimmt, ob das vom Benutzer eingegebene Jahr ein Schaltjahr ist. Dies soll mithilfe zweier Funktionen erfolgen.
Zuerst soll in der Funktion `bekomme_jahr()` ein Jahr vom Benutzer eingegeben werden und als Ganzzahl zurückgegeben werden.s
In der Funktion `pruefe_schaltjahr(jahr)` soll dann geprüft werden, ob das übergebene Jahr ein Schaltjahr ist. Ist es ein Schaltjahr, soll der `boolean True` zurückgegeben werden, andernfalls `False`.
Prüfen Sie am Ende mit dem unten stehenden Code, ob Ihre Funktionen funktionieren.

Funktion für die Jahr-Eingabe:

```
def bekomme_jahr():
    pass
```

Funktion für die Schaltjahr-Prüfung:

Zusatzinfo:
Ist ein Jahr durch 4 teilbar, ist es ein Schaltjahr
- Kann man es zusätzlich durch 100 teilen, ist es kein Schaltjahr
- Kann man es zusätzlich durch 100 und 400 teilen, ist es ein Schaltjahr

Alle anderen Jahre sind keine Schaltjahre

```
def pruefe_schaltjahr(jahr):
    pass
```

Prüfen Sie Ihre Funktionen durch Ausführen des folgenden Codes:s

```
user_input = bekomme_jahr() # Speichert das eingegebene Jahr als
Variable ab

if pruefe_schaltjahr(user_input): # Falls das Jahr ein Schaltjahr
ist
    print(f"{user_input} ist ein Schaltjahr")

else: # Falls das Jahr kein Schaltjahr ist
    print(f"{user_input} ist kein Schaltjahr")
```

Versuchen Sie es erneut. Geben Sie eine Jahreszahl ein: 1200

Bonus:
- Passen Sie die Funktion bekomme_jahr() so an, dass die Eingabe nur zurückgegeben wird, wenn es sich um eine Ganzzahl handelt
- Ist dies nicht der Fall, soll solange erneut nach einem Jahr gefragt werden, bis der Benutzer eine Ganzzahl eingibt

Aufgabe B: Trading Algorithmus

Ein Finanz-Startup beauftragt Sie einen Trading-Bot für Aktien zu programmieren. Dieser soll anhand von Aktienwerten der letzten 7 Tage entscheiden, ob eine Aktie gekauft oder verkauft werden soll.
Dazu stellt das Startup folgende *Strategien* auf: (fiktiv, keine Handlungsempfehlung)
- Ist der aktuellste Wert im Vergleich zum ältesten Wert gesunken, so soll verkauft werden, andernfalls gekauft
- Liegt der aktuellste Wert unter dem Durchschnittswert, soll gekauft werden, andernfalls verkauft

Schreiben Sie für jede Strategie eine Funktion, die zurückgibt (einfaches `print()` reicht), ob gekauft oder verkauft werden soll. Grundlage bzw. Input für die Funktion soll eine Liste [10, 12, 9, ...] mit 7 Elementen sein, bei der das letze Element den aktuellsten Wert, und das erste Element den ältesten Wert repräsentiert. Testen Sie Ihre Funktionen mit den vorgegebenen Listen.
Wichtig: Der aktuellste Wert soll bei der Berechnung zum Durchschnittswert etc. nicht berücksichtigt werden

```
apple = [160, 166, 165, 166, 171, 169, 170]

basf = [60, 56, 57, 49, 52, 51, 50]

alibaba = [110, 120, 113, 118, 115, 116, 117]S
```

Aufgabe C: Palindrom

Schreiben Sie eine Funktion, die mithilfe einer Schleife prüft, ob ein Wort ein Palindrom (Wort, das man von vorne gleich wie von hinten lesen kann) ist, oder nicht. Tipp: Setzen sie vor der Überprüfung das Wort in Klein- oder Großbuchstaben
Testen Sie ihr Programm z.B. mit den folgenden Wörtern:

```
wort1 = "Uhu"
wort2 = "Legovogel"
wort3 = "Universität"
wort4 = "Lagerregal"
```