4. Fortgeschrittene Programmierkonzepte Teil 2

4.1. map()-Funktion

- mit der Funktion **map()** lässt sich eine Funktion in einer Anweisung mehrfach mit unterschiedlichen Werten aufrufen
    - Anzahl der Parameter muss bei der Definition und dem Aufruf übereinstimmen

```
# Definition der Funktion zum Verdoppeln eines Wertes
def verdoppeln(wert):
    """
    :param wert:
    :return:
    """
    return wert * 2

# für die Werte in der Liste ist das Doppelte zu berechnen
liste = [12, 2, 56]
# Schleife zum Abarbeiten der Elemente in der Liste
for i in liste:
    print(verdoppeln(i))


# map()-Funktion anstelle der Schleife
# die Funktion wird für alle Elemente der Liste aufgerufen
ergebnis = map(verdoppeln, liste)
for i in ergebnis:
    print(i)
```

4.2. Anonyme Funktionen (lambda) für iterierbare Objekte

- mit der Funktion map() lässt sich eine Funktion – hier ein lambda-Ausdruck - in einer Anweisung mehrfach mit unterschiedlichen Werten aufrufen
- Gegenüberstellung einer benannten Funktion und ein lambda-Ausdruck:

```
# Definition einer benannten Funktion
def symbol (parameterliste):
    return rückgabewert

# Definition eines lambda-Ausdrucks
symbol = lambda Parameterliste: rückgabewert
```

- Berechnung für die Elemente einer Liste mit **map()** mit lambda-Ausdruck:

```
# Definiton lambda-Ausdruck für das Verdoppeln eines Wertes
# map()-Funktion anstelle der Schleife
# die Funktion wird für alle Elemente der Liste aufgerufen
# die Liste der berechneten Werte ausgeben
print(list(map(lambda x: x*2, liste)))
```

- Berechnung für die variable Anzahl von Elementen bei einem lambda-Ausdruck

```
average = lambda *args: sum(args) / len(args)
print(average(1, 2, 3, 4, 5))
```

4.3. Listen-Abstraktion (List Comprehension)

- spezielle Anweisungen zum Erzeugen von generischen Inhalten für Datentypen wie Listen und Dictionaries
- effektive und kompakte Möglichkeit, mit wenig Code aus iterierbaren Objekten (Listen oder Dictionaries) andere iterierbare Objekte zu erstellen
- Ermittle alle positiven Werte aus einer Liste mit positiven und negativen Werten

```
zahlen = [2.34, -56.6, -12.4, 67.9]
pos_zahlen = [x for x in zahlen if x > 0]
print(pos_zahlen)
```

- Ermittle die Vokale in einem Text aus mehreren Worten

```
satz = "Python ist eine Programmiersprache"
vokale = {i for i in satz if i in 'aeiou'}
print(vokale)
```

4.4. filter()-Funktion

- mit der Funktion **filter()** lässt ein iterierbares Objekt (Liste oder Dictionary) mit Hilfe einer Funktion untersuchen; sie liefert diejenigen Elemente, für die die Funktion True ermittelt:

```
# Definition der Funktion zum Prüfen
def positiv(wert):
    """
    :param wert:
    :return:
    """
    if wert > 0:
        t = True
    else:
        t = False
    return t

# aus diesen Elementen in der Liste sind die pos. Werte zu ermitteln
pos_liste = filter(positiv, [12, 2,-56, 0, 3, -5])
# Schleife zum Ausgeben der Elemente in der Liste
for i in pos_liste:
    print(i)
```

Aufgaben

1. Umrechnung Fahrenheit in Celsius
   für die Umrechnung von Fahrenheit in Celsius gilt: $\mathbf{C} = (°\mathbf{F} - 32) * \frac{5}{9}$
   Für die Programmierung in Python soll ein Lambda-Ausdruck definiert werden.
   Die Temperaturwerte [°C] als Elemente in einer Liste sollen über eine Schleife abgearbeitet und die Temperaturwerte in °F ermittelt und auf die Konsole ausgegeben werden.

2. Umrechnung Celsius in Fahrenheit
   für die Umrechnung von Celsius in Fahrenheit gilt: $°\mathbf{F} = \mathbf{C} * \frac{9}{5} + 32$
   Für die Programmierung in Python soll ein Lambda-Ausdruck definiert werden.
   Die Temperaturwerte [°F] als Elemente in einer Liste sollen über eine Schleife abgearbeitet und die Temperaturwerte in °C ermittelt und auf die Konsole ausgegeben werden.

3. Aufgabe 1 mit Listenabstraktion (List Comprehension) lösen

4. Aufgabe 2 mit Listenabstraktion (List Comprehension) lösen