Programmierung in Python

Univ.-Prof. Dr. Martin Hepp, Universität der Bundeswehr München

Einheit 2: Kontrollflussstrukturen und Algorithmik

Version: 2019-11-08

1 Sequenz

In [12]:
```
print('Erste Anweisung')
print('Zweite Anweisung')
print('Dritte Anweisung')
```

```
Erste Anweisung
Zweite Anweisung
Dritte Anweisung
```

Univ.-Prof. Dr. Martin Hepp, martin.hepp@unibw.de

2 Schleifen

Univ.-Prof. Dr. Martin Hepp, martin.hepp@unibw.de

2 Schleifen

2.1 Grundlagen: Schleifen mit `for`

In [3]:
```
# Schleife für Liste mit Zahlen
zahlenreihe = [1, 2, 3, 4]
for zahl in zahlenreihe:
    print(zahl)
```

```
1
2
3
4
```

Univ.-Prof. Dr. Martin Hepp, martin.hepp@unibw.de

Schleife über Liste mit Zeichenketten:

In [4]:
```
freunde = ['Peter', 'Paul', 'Mary']
for person in freunde:
    print('Hallo, ' + person + '!')
```

```
Hallo, Peter!
Hallo, Paul!
Hallo, Mary!
```

Schleife über Zeichen in Zeichenkette

Univ.-Prof. Dr. Martin Hepp, martin.hepp@unibw.de

In [2]:
```
text = 'UniBwM'
for char in text:
    print(char)
```

```
U
n
i
B
w
M
```

Mehrere Anweisungen innerhalb einer Schleife:

Die Einrückung bestimmt, dass der Befehl zum Schleifeninhalt gehört.

In [17]:
```
for zahl in [1, 2, 3, 4]:
    print(zahl)
    print('Hallo')
```

```
1
Hallo
2
Hallo
3
Hallo
4
Hallo
```

Was passiert hier?

In [18]:
```
for zahl in [1, 2, 3, 4]:
    print(zahl)
print('Hallo')
```

```
1
2
3
4
Hallo
```

2.2 `range` als Iterable

Manchmal möchte man eine Schleife auf eine bestimmte Folge an Zahlen anwenden.

`range()` ist eine Hilfsfunktion, mit er ein Hilfsobjekt erzeugt werden kann, das eine bestimte Folge an Ganzzahlen liefert, über die eine Schleife ablaufen soll.

In [3]:
```
for zahl in range(5):
    print(zahl)
```

```
0
1
2
3
4
```

2.2.1 Anfangswert und obere Schranke

Wenn zwei Parameter angegeben werden, ist der erste Wert die erste zu erzeugende Zahl und das zweite die obere Schranke, die gerade **nicht mehr** enthalten sein soll.

In [4]:
```
for zahl in range(4, 10):
    print(zahl)
```

```
4
5
6
7
8
9
```

2.2.2 Schrittweite

Durch Angabe eines dritten Parameters kann die Schrittweite spezifiziert werden. So wird z.B. nur jede zweite Zahl von 0 bis 7 erzeugt:

In [9]:
```
for zahl in range(0, 8, 2):
    print(zahl)
```

```
0
2
4
6
```

2.2.2 Schrittweite

Durch Angabe eines dritten Parameters kann die Schrittweite spezifiziert werden. So wird z.B. nur jede zweite Zahl von 0 bis 7 erzeugt:

In [9]:
```
for zahl in range(0, 8, 2):
    print(zahl)
```

```
0
2
4
6
```

Das funktioniert auch mit negativen Schrittweiten. Dann allerdings muss man darauf achten, dass **der Startwert (hier 5) größer ist als die obere Schranke (hier 0).**

2.2.3 Schleifen mit einer nicht-ganzzahligen Schrittweite

Manchmal möchte man eine Schleife über Zahlenfolgen ausführen, die eine Schrittweite verwenden, die nicht ganzzahlig ist, z.B. die Folge $[1,0; 1,1; 1.2]$.

`range()` funktioniert nur für ganze Zahlen.

Es gibt drei Wege, wie man dies umgehen kann:

2.2.3.1 Skalierung mit einem Faktor

Man kann einfach die gewünschte Zahlenfolge um einen festen Faktor vergrößern und den Wert dann im Innern der Schleife ggfls.

2.2.3.2 Erzeugen einer Liste aus Gleitkommazahlen

Man kann die Liste, über die die Schleife ablaufen soll, auch vorab manuell erzeugen.

Beispiel:

In [50]:
```
zahlen = []
for i in range(10, 13):
    zahlen.append(i/10)
print(zahlen)
```

```
[1.0, 1.1, 1.2]
```

In [51]:
```
for zahl in zahlen:
    print(zahl)
```

```
1.0
1.1
1.2
```

2.2.3.2 Erzeugen einer Liste aus Gleitkommazahlen

Man kann die Liste, über die die Schleife ablaufen soll, auch vorab manuell erzeugen.

Beispiel:

In [50]:
```
zahlen = []
for i in range(10, 13):
    zahlen.append(i/10)
print(zahlen)
```

```
[1.0, 1.1, 1.2]
```

In [51]:
```
for zahl in zahlen:
    print(zahl)
```

```
1.0
1.1
1.2
```

2.2.3.3 Spezialfunktionen aus der Bibliothek `numpy`

In der Bibliothek `numpy` gibt es spezielle Funktionen, um Zahlenfolgen aus nicht-ganzzahligen Werten zu erzeugen.

Diese sind nicht Gegenstand der Vorlesung und werden hier nur der Vollständigkeit halber genannt.

- numpy.arange
- numpy.linspace

2.3 Übung

Geben Sie alle Zweierpotenzen von 2^0 bis 2^7 aus.

2.3 Übung

Geben Sie alle Zweierpotenzen von 2^0 bis 2^7 aus.

In [14]:
```
for exponent in range(0, 8):
    print('2 hoch', exponent, 'ist gleich', 2**exponent)
```

```
2 hoch 0 ist gleich 1
2 hoch 1 ist gleich 2
2 hoch 2 ist gleich 4
2 hoch 3 ist gleich 8
2 hoch 4 ist gleich 16
2 hoch 5 ist gleich 32
2 hoch 6 ist gleich 64
2 hoch 7 ist gleich 128
```

2.4 Schleifen mit `while`

In [19]:
```
obere_schranke = 100
aktueller_wert = 1
while aktueller_wert < obere_schranke:
    print(aktueller_wert)
    aktueller_wert = aktueller_wert * 2
```

```
1
2
4
8
16
32
64
```

Univ.-Prof. Dr. Martin Hepp, martin.hepp@unibw.de

2.4.1 Schleifenabbruch mit `break`

Man kann eine while-Schleife mit der Anweisung `break` verlassen:

In [47]:
```
a = 0
while a < 10:
    a = a + 1
    print(a)
    if a == 7:
        print('7 erreicht, beende die Schleife')
        break
```

```
1
2
3
4
5
6
7
7 erreicht, beende die Schleife
```

3 Verzweigungen

3.1 Einfache Verzweigung mit `if`

In [59]:
```
wert = input('Geben Sie eine Zahl ein. ')
wert = int(wert)
if wert > 0:
    print(wert, 'ist größer als Null.')
```

```
Geben Sie eine Zahl ein. 4
4 ist größer als Null.
```

3.2 Verzweigung mit `if` und `else`

In [60]:
```
wert = input('Geben Sie eine Zahl ein. ')
wert = int(wert)
if wert > 0:
    print(wert, 'ist größer als Null.')
else:
    print(wert, 'ist kleiner oder gleich Null.')
```

```
Geben Sie eine Zahl ein. 4
4 ist größer als Null.
```

3.3 Mehrfachverzweigung mit `if`, `elif` und `else`

In [61]:
```
wert = input('Geben Sie eine Zahl ein. ')
wert = int(wert)
if wert > 0:
    print(wert, 'ist größer als Null.')
elif wert == 0:
    print(wert, 'ist gleich Null.')
else:
    print(wert, 'ist kleiner als Null.')
```

```
Geben Sie eine Zahl ein. -3
-3 ist kleiner als Null.
```

3.4 Fallstricke bei Mehrfachverzweigungen

Die Bedingungen müssen vom spezielleren Fall zu den allgemeineren Fällen geprüft werden, da nur der erste passende Zweig ausgeführt wird.

In [43]:
```
zahl = 4
if zahl > 0:
    print('Die Zahl ist größer als Null.')
elif zahl > 2:
    print('Die Zahl größer als 2.')
```

```
Die Zahl ist größer als Null.
```

3.4 Fallstricke bei Mehrfachverzweigungen

Die Bedingungen müssen vom spezielleren Fall zu den allgemeineren Fällen geprüft werden, da nur der erste passende Zweig ausgeführt wird.

In [43]:
```
zahl = 4
if zahl > 0:
    print('Die Zahl ist größer als Null.')
elif zahl > 2:
    print('Die Zahl größer als 2.')
```

```
Die Zahl ist größer als Null.
```

Die zweite Prüfung `elif zahl > 2:` wird nicht mehr ausgeführt, weil die Bedingung für den ersten Zweig `if zahl > 0:` bereits erfüllt ist.

3.5 Test, ob Wert innerhalb eines Intervalls liegt

Oft muss man prüfen, ob ein Wert innerhalb eines Intervals liegt.

Beispiel: $0 < x < 10$

In Python gibt es zwei Wege, dies auszudrücken:

In [56]:
```
x = 5
# Logische Kombination mehrerer Vergleiche mit 'and'
if 0 < x and x < 10:
    print("0 < x < 10")
```

```
0 < x < 10
```

3.5 Test, ob Wert innerhalb eines Intervalls liegt

Oft muss man prüfen, ob ein Wert innerhalb eines Intervals liegt.

Beispiel: $0 < x < 10$

In Python gibt es zwei Wege, dies auszudrücken:

In [56]:
```
x = 5
# Logische Kombination mehrerer Vergleiche mit 'and'
if 0 < x and x < 10:
    print("0 < x < 10")
```

```
0 < x < 10
```

In [58]:
```
x = 3
# Direkter Ausdruck
if 0 < x < 10:
```

4 Kombinationen

4.1 Schleifen in Schleifen

In [25]:
```
for faktor_1 in range(1, 11):
    for faktor_2 in range(1, 11):
        # end  = '\t' sorgt dafür, dass am Ende des Befehls
        # keine neue Zeile, sondern ein Tabulatorzeichen folgt.
        print(faktor_1 * faktor_2, end = '\t')
    print()
```

```
1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
```

4.2 Verzweigungen in Schleifen

In [26]:
```
for zahl in [0, 1, 2, 3, 4, 5]:
    # zahl % 2 liefert den Divisionsrest bei einer ganzzahligen Division
    if zahl % 2 == 0:
        print(zahl, ' ist gerade')
    else:
        print(zahl, ' ist ungerade')
```

```
0  ist gerade
1  ist ungerade
2  ist gerade
3  ist ungerade
4  ist gerade
5  ist ungerade
```

4.3 Übung

Ein Sparkonto soll fünf Jahre lang jeden Monat mit 1 % verzinst werden. Am Ende jedes Jahres wird eine Kontoführungsgebühr von 5 EUR abgezogen.

4.3.1 Aufgabe

In [62]:
```
anzahl_jahre = 5
zinssatz = 0.01  # pro Monat
guthaben = 1200.0

# Hier einfügen
```

4.3.2 Musterlösung

In [63]:
```
for jahr in range(anzahl_jahre):
    print(jahr, end = ':\t')
    for monat in range(12):
        guthaben = guthaben * (1 + zinssatz)
        print(round(guthaben, 2), end = '\t')
    guthaben = guthaben - 5
    print(jahr, round(guthaben, 2))
```

```
0:      1212.0  1224.12 1236.36 1248.72 1261.21 1273.82 1286.56 1299.43 1312.42
1325.55 1338.8  1352.19 0 1347.19
1:      1360.66 1374.27 1388.01 1401.89 1415.91 1430.07 1444.37 1458.81 1473.4
1488.14 1503.02 1518.05 1 1513.05
2:      1528.18 1543.46 1558.89 1574.48 1590.23 1606.13 1622.19 1638.41 1654.8
1671.35 1688.06 1704.94 2 1699.94
3:      1716.94 1734.11 1751.45 1768.96 1786.65 1804.52 1822.57 1840.79 1859.2
1877.79 1896.57 1915.53 3 1910.53
4:      1929.64 1948.94 1968.43 1988.11 2007.99 2028.07 2048.35 2068.84 2089.52
2110.42 2131.52 2152.84 4 2147.84
```

Hinweise:

- Die Rundung erfolgt hier nur bei der Ausgabe!
- Für die Rundung bei der Ausgabe gibt es spezielle Mechanismen, die hier aber nicht behandelt werden.
- Man muss festlegen, ob die Gebühr nach oder vor der Zinsgutschrift für den letzten Monat eines Jahres erfolgen soll.

Vielen Dank!

http://www.ebusiness-unibw.org/wiki/Teaching/PIP

Univ.-Prof. Dr. Martin Hepp, martin.hepp@unibw.de