Programmierung in Python

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

Einheit 2: Kontrollflussstrukturen und Algorithmik

Version: 2019-12-05

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

1 Kontrollflussstrukturen

Der Kontrollfluss eines Programms definiert, in welcher Reihenfolge, wie oft und unter welchen Bedingungen einzelne Befehle ausgeführt werden.

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

2 Sequenz

Eine Sequenz ist die einfachste Kontrollflussstruktur. Bei einer Sequenz werden einzelne Anweisungen einfach nacheinander ausgeführt.

In [2]:
```
print('Erste Anweisung')
print('Zweite Anweisung')
print('Dritte Anweisung')
print('Vierte Anweisung')
```

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

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

3 Schleifen

Eine Schleife ist eine Kontrollflusstruktur, bei der eine Anweisung oder mehrere Anweisungen wiederholt werden. Hier gibt es mehrere Varianten.

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

3.1 Grundlagen: Schleifen mit `for`

Eine Schleife, die über eine gegebene Folge von Objekten ablaufen soll, wird in Python mit `for` realisiert.

Die Syntax ist dabei

```
for <arbeitsvariable> in <eingangsdaten>:
    befehl ...
```

Die Schleife wird so oft ausgeführt, wie ***eingangsdaten*** Unterlemente hat. Bei jedem Durchgang ist der aktuelle Wert des aktuell bearbeiteten Unterelementes über den Namen von ***arbeitsvariable*** zugänglich.

```
In [29]: for zahl in [1, 3, 2, 4]:
```

3.1 Grundlagen: Schleifen mit `for`

Eine Schleife, die über eine gegebene Folge von Objekten ablaufen soll, wird in Python mit `for` realisiert.

Die Syntax ist dabei

```
for <arbeitsvariable> in <eingangsdaten>:
    befehl ...
```

Die Schleife wird so oft ausgeführt, wie ***eingangsdaten*** Unterlemente hat. Bei jedem Durchgang ist der aktuelle Wert des aktuell bearbeiteten Unterelementes über den Namen von ***arbeitsvariable*** zugänglich.

In [29]:
```
for zahl in [1, 3, 2, 4]:
```

Schleife über Liste mit Zeichenketten:

In [17]:
```
freunde = ['Peter', 'Paul', 'Mary']
for person in freunde:
    print('Hallo, ' + person + '!')
```

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

Schleife über Zeichen in Zeichenkette

In [18]:
```
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
```

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
```

In [ ]:
```
# Entspricht
zahlenfolge = [1, 2, 3, 4]
zahl = zahlenfolge[0]
```

Was passiert hier?

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

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

3.2 `range` als Iterable

Manchmal möchte man eine Schleife auf eine bestimmte Folge an Zahlen anwenden. `range()` ist eine Hilfsfunktion, mit er ein Hilfsobjekt erzeugt wird, das eine bestimte Folge an Ganzzahlen liefert, über die eine Schleife ablaufen soll.

In [3]:
```
# Alle Zahlen von 0 bis 4
for zahl in range(5):
    print(zahl)
```

```
0
1
2
3
4
```

3.2.1 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
```

3.2.2 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
```

Negative Schrittweite:

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).**

In [48]:
```
for zahl in range(5, 0, -1):
    print(zahl)
```

```
5
4
3
2
1
```

3.2.3 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:

3.2.3.1 Skalierung mit einem Faktor

Man kann einfach die gewünschte Zahlenfolge um einen festen Faktor vergrößern und den Wert dann im Innern der Schleife ggfls. durch diesen Faktor teilen. Im Beispiel würde man also über `[10, 11, 12]` iterieren:

In [49]:
```
for i in range(10, 13):
    print(i/10)
```

```
1.0
1.1
1.2
```

3.2.3.2 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]
```

3.2.3.2 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
```

In der Praxis verwendet man dafür sogenannte *List Comprehensions*, eine Kurzform, um Listen über eine Schleife zu erstellen. Das Beispiel von oben sähe dann so aus:

In [53]:
```
zahlen = [i/10 for i in range(10, 13)]
print(zahlen)
```

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

3.2.3.3 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

3.3 Übung

Schreiben Sie ein Programm, das mit Hilfe einer `for`-Schleife alle Zweierpotenzen von 2^0 bis 2^7 ausgibt.

3.3 Übung

Schreiben Sie ein Programm, das mit Hilfe einer `for`-Schleife alle Zweierpotenzen von 2^0 bis 2^7 ausgibt.

In [22]:
```
for exponent in range(0, 8):
    print(2**exponent)
```

```
1
2
4
8
16
32
64
128
```

3.4 Schleifen mit `while`

Wenn eine Schleife so oft wiederholt werden soll, bis eine Bedingung erfüllt ist, oder solange eine Bedingung nicht erfüllt ist, verwendet man besser eine Schleife 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
```

3.4.1 Schleifenabbruch mit `break`

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

In [31]:
```
a = 0
while a < 5:
    a = a + 1
    print(a)
    if a == 3:
        print('3 erreicht, beende die Schleife')
        break
```

```
1
2
3
3 erreicht, beende die Schleife
```

3.4.2 Ausführung von Anweisungen, wenn die Schleife ohne `break` beendet wurde

Manchmal ist es nützlich, wenn man Anweisungen nur dann ausführt, wenn eine `while`-Schleife regulär beendet wurde, also nicht über eine `break`-Anweisung verlassen wird. Dazu dient das Schlüsselwort `else`, das **auf der selben Einrückungsebene wie der `while`-Befehl stehen muss.**

Eine typische Anwendung ist eine Meldung bei einem erfolglosen Versuch:

Beispiel für `else` am Ende einer `while`-Schleife:

In [67]:
```
antwort = False  ## nur ein Dummy

versuche = 3
while versuche > 0:
    print('Hallo, bitte melde Dich!')
    if antwort == True:
        # Jemand hat sich gemeldet
        break
    versuche = versuche - 1
else:
    print('Abbruch - keine Antwort.')
```

```
Hallo, bitte melde Dich!
Hallo, bitte melde Dich!
Hallo, bitte melde Dich!
Abbruch - keine Antwort.
```

4 Verzweigungen

Verzweigungen werden verwendet, um einzelne Befehle nur dann auszuführen, wenn eine bestimmte Bedingung erfüllt ist. Sie entsprechen Fallunterscheidungen in der Mathematik.

4.1 Einfache Verzweigung mit `if`

In [25]:
```
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
```

4.2 Verzweigung mit `if` und `else`

In [27]:
```
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. -3
-3 ist kleiner oder gleich Null.
```

4.3 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.
```

4.4 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.
```

4.4 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.

4.5 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:

4.5 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 [ ]:
```
x = 5
# Logische Kombination mit 'and'
if 0 < x and x < 10:
    print("0 < x < 10")
```

4.5 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 [ ]:
```
x = 5
# Logische Kombination mit 'and'
if 0 < x and x < 10:
    print("0 < x < 10")
```

In [ ]:
```
x = 3
# Direkter Ausdruck
if 0 < x < 10:
    print("0 < x < 10")
```

5 Kombinationen

Richtige Programme entstehen dadurch, dass man Sequenzen, Schleifen und Verzweigungen kombiniert, insbesondere ineinander verschachtelt.

5.1 Schleifen in Schleifen

In [33]:
```
for faktor_1 in range(1, 4):
    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	
```

5.2 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
```

5.3 Übung

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

5.3.1 Aufgabe

In [35]:
```
anzahl_jahre = 5
zinssatz = 0.01  # pro Monat
guthaben = 1200.0

# Hier einfügen
```

5.3.2 Musterlösung

In [39]:
```
for jahr in range(anzahl_jahre):
    print(jahr, end = ':\t')
    for monat in range(12):
        guthaben = guthaben * (1 + zinssatz)
        print(f'{guthaben:7.2f}', end = '\t')
    guthaben = guthaben - 5
    print()
    print(f'Saldo am Ende von Jahr {jahr}: {guthaben:7.2f}')
```

```
0:      3908.47 3947.56 3987.03 4026.90 4067.17 4107.84 4148.92 4190.41 4232.31
4274.64 4317.38 4360.56
Saldo am Ende von Jahr 0: 4355.56
1:      4399.11 4443.10 4487.53 4532.41 4577.73 4623.51 4669.75 4716.44 4763.61
4811.24 4859.36 4907.95
Saldo am Ende von Jahr 1: 4902.95
2:      4951.98 5001.50 5051.51 5102.03 5153.05 5204.58 5256.63 5309.19 5362.28
5415.91 5470.07 5524.77
Saldo am Ende von Jahr 2: 5519.77
3:      5574.97 5630.71 5687.02 5743.89 5801.33 5859.34 5917.94 5977.12 6036.89
6097.26 6158.23 6219.81
```

6 Quellenangaben und weiterführende Literatur

[Pyt2019] Python Software Foundation. Python 3.8.0 Documentation. https://docs.python.org/3/.

Vielen Dank!

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

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